FELICIA IONESCU

Inginerie electronică

GRAFICA

în realitatea virtuală

Editura Tehnică

Colecţia Inginerie electronică

Coordonatorul colecţiei: **prof. univ. dr. ing. Mircea BODEA**

Editura TEHNICĂ îşi propune să găzduiască în cadrul colecţiei ***Inginerie electronică*** cărţi fundamentale, care şi-au demonstrat perenitatea prin nivelul deosebit de competitiv al abordării didactice şi ştiinţifice a subiectului, dar şi prin contribuţia la satisfacerea cererii în continuă creştere de proiectanţi de circuite şi de sisteme microelectronice analogice, digitale sau semnale mixte.

Ca urmare, proiectarea de circuite şi sisteme microelectronice, prin implicaţiile sale ştiinţifice şi economice, este un subiect de o importanţă majoră care se regăseşte rezonant atât în zona de interes a instruirii universitare şi postuniversitare, cât şi în aceea a interesului profesional al celor implicaţi în realizarea de sisteme microelectronice, indiferent de faptul că sunt juniori, seniori sau veterani.

Dată fiind „geometria variabilă" a domeniului de inginerie electronică şi dinamica sa deosebit de accentuată, subiectele cărţilor acestei colecţii se vor concentra îndeosebi pe fundamente, îmbinând abordarea didactică cu aceea de tip ingineresc. În acest fel, seria va acoperi atât cerinţele ingineriei incrementale, în care dezvoltarea se face prin contiguitate, cât şi pe cele ale ingineriei fundamentale, în care caracteristica majoră a dezvoltării o constituie discontinuitatea, ruperea de ritm.

În această colecţie vor apărea:

- Gheorghe Ştefan **Circuite integrate digitale**
- Corneliu Burileanu **Microprocesoare – arhitecturi CISC şi RISC**

Prof. univ. **Felicia IONESCU**

GRAFICA ÎN REALITATEA VIRTUALĂ

EDITURA TEHNICĂ
Bucureşti, 2000

Prefață

Realitatea virtuală este o tehnologie care a modificat şi continuă să modifice modul de abordare a numeroase aplicaţii practice în industrie, medicină, instruire, artă şi divertisment. Deşi începuturile realităţii virtuale se pot situa la nivelul anilor 1965-1970, legate de cercetările lui Ivan Sutherland şi de realizarea unor simulatoare de antrenament pentru piloţi, cercetări sistematizate în domeniul realităţii virtuale au avut loc mai cu seamă în ultimii 10-15 ani.

Realitatea virtuală înglobează o mare varietate de tehnologii şi echipamente (echipamente electronice, hidraulice, sisteme de calcul în timp real, proiectoare optice), care generează senzaţii adresate simţurilor umane (simţului vizual, auditiv, tactil). Dintre acestea, simţul vizual este cel mai intens folosit în realitatea virtuală, deoarece cele mai multe informaţii despre mediul înconjurător se obţin prin intermediul văzului. Imaginea vizuală este componenta esenţială a realităţii virtuale, fiind aceea care permite utilizatorului să identifice mediul în care evoluează, să se orienteze şi să acţioneze şi, de aceea, cele mai multe cercetări în domeniul realităţii virtuale au fost făcute în domeniul graficii şi al generării imaginilor.

Grafica utilizată în realitatea virtuală permite generarea prin calcul a imaginilor, pornind de la modelele obiectelor tridimensionale care alcătuiesc scena virtual. Două cerinţe de performanţă ale graficii sunt vitale în sistemele de realitate virtuală: realismul imaginilor şi generarea acestora în timp real. Aceste cerinţe implică atât aspecte software, de selectare a celor mai adecvaţi algoritmi de generare a imaginilor, cât şi aspecte hardware, de realizare a unor echipamente performante, care să asigure realismul şi viteza de generare a imaginilor în realitatea virtuală. Staţii grafice multiprocesor şi acceleratoare grafice care implementează în hardware algoritmii de generare a imaginilor sunt echipamente de bază în realitatea virtuală, iar progresele tehnologice realizate au permis utilizarea acestora într-un număr imens de aplicaţii de realitate virtuală, accesibile în momentul de faţă unor categorii variate de utilizatori.

Realitatea virtuală este doar una dintre aplicaţiile graficii pe calculator, alături de altele cum sunt: realizarea interfeţelor utilizator dezvoltate în numeroase programe utilitare şi medii de programare, proiectarea asistată de calculator (CAD – *Computer Aided Design*), prezentările grafice interactive, vizualizarea datelor ştiinţifice, tehnologia multimedia.

În această lucrare se pune accentul pe conexiunea dintre aspectele teoretice ale graficii şi implementarea programelor de generare a imaginii obiectelor şi scenelor virtuale tridimensionale. În primele capitole sunt prezentate operaţiile de bază în grafica pe calculator: modelarea obiectelor tridimensionale, transformări

geometrice în spațiu, sisteme de vizualizare, transformarea de rastru. În cea de-a doua parte a lucrării se reiau toate aceste aspecte ale generării imaginilor din perspectiva programării, folosind biblioteca grafică OpenGL, sistemul de dezvoltare GLUT şi limbajul de modelare în realitatea virtuală VRML. De asemenea, sunt prezentate aspectele avansate ale generării imaginilor tridimensionale (anti-aliasing, umbrire, texturare), atât din punct de vedere teoretic, cât şi al abordării în programare.

Unul din scopuri principale ale cărții este de a oferi soluții de implementare a aplicațiilor grafice, în special cele necesare în realitatea virtuală, prin prezentarea metodelor actuale şi accesibile de rezolvare a fiecărei etape a procesului de generare a imaginilor. În acest sens, aproape toate imaginile ilustrative ale diferitelor tehnici au fost produse de autoare prin programe grafice descrise în text. Cea mai mare parte a exemplelor prezentate se bazează pe experiența de implementare a unor sisteme grafice integrate în simulatoare de zbor, realizate în colaborare cu cercetătorii şi inginerii de la Institutul de Simulatoare Simultec SA, Bucureşti. De asemenea, materialul prezentat are ca suport experiența de curs *Grafică şi Realitate Virtuală*, predat studenților anului IV de la Facultatea de Electronică şi Telecomunicații, Universitatea Politehnica Bucureşti.

Pentru realizarea lucrării am avut sprijinul permanent al mai multor colegi şi al colectivului Laboratorului de Ingineria Informației din Catedra de Electronică Aplicată şi Ingineria Informației. Regretatul prof. dr. ing. Adrian-Traian Murgan a fost cel care a introdus disciplina *Grafica în Realitatea Virtuală* la secția de Ingineria Informației din Facultatea de Electronică şi Telecomunicații şi a sprijinit activitatea în acest domeniu. Preparatorii Andrei Jalbă şi Bogdan Popescu au realizat o parte importantă din implementările algoritmilor de generare a imaginilor.

Cartea se adresează în primul rând studenților din domeniul sistemelor de calcul şi ingineria informației, fiind totodată utilă şi inginerilor şi proiectanților de aplicații grafice în realitatea virtuală sau în alte domenii. Indexul de termeni prezintă pentru fiecare denumire versiunea în limba engleză, ca bază comună de interpretare a termenilor, sau ca singura denumire pentru acei termeni a căror traducere este prea puțin intuitivă.

În domeniul graficii în realitatea virtuală, dezvoltările tehnologice sunt deosebit de dinamice: apar în permanență echipamente din ce în ce mai performante, la prețuri din ce în ce mai scăzute; apar numeroase biblioteci, limbaje şi programe utilitare (toolkits) care propun diferite modalități de abordare a aplicațiilor grafice. Din acest motiv, în lucrare s-a insistat mai mult pe aspectele teoretice ale graficii şi realității virtuale, şi mai puțin pe descrieri tehnologice care pot fi depăşite într-un timp foarte scurt.

Aprilie, 2000

Autoarea

CUPRINS

1

INTRODUCERE ÎN REALITATEA VIRTUALĂ

Realitatea Virtuală a cunoscut în ultimii ani o imensă dezvoltare şi publicitate. În reviste, magazine, la televiziune, s-a prezentat această "nouă şi fantastică tehnologie" din cele mai variate puncte de vedere. Chiar şi definiţiile date termenului *Realitate Virtuală* diferă de la un autor la altul, una din definiţiile cele mai răspândite fiind următoarea: *un sistem de Realitate Virtuală este un sistem care creează unui utilizator impresia că este "imersat"(prezent) într-un mediu sintetic* [Gig93]. Termenul *Realitate Virtuală* (*Virtual Reality*) este unul dintre termenii folosiţi pentru a defini astfel de experimente, dar se mai folosesc şi termenii: *lume virtuală* (*Virtual World*), *mediu virtual (Virtual Environment*), *mediu sintetic* (*Synthetic Environment*), *realitate artificială (Artificial Reality*). Deşi aceste denumiri sunt echivalente, unii specialişti le preferă, deoarece termenul de *Realitate Virtuală*, a fost şi este suprautilizat şi, de multe ori, asociat cu aşteptări lipsite de realism. Termenul *Cyberspace*, care se mai foloseşte pentru a desemna un mediu virtual, este, oarecum, nepotrivit, deoarece este preluat din denumirea folosită de William Gibson în romanul de ficţiune "*The Matrix*", pentru descrierea unei lumi a viitorului, tehno-criminală, invadată de droguri, implantări cerebrale şi furturi prin calculator. Este normal ca specialişti care muncesc din greu pentru progrese în domeniul realităţii virtuale să nu dorească asocierea acestui domeniu cu previziuni atât de sumbre.

Cuvântul "virtual" este folosit în mod obişnuit în domeniul calculatoarelor pentru a desemna o entitate care simulează o altă entitate. De exemplu, termenul "memorie virtuală" se referă la simularea memoriei principale prin memoria hard-discului. Sistemul de operare stochează pagini de memorie în fişiere pe disc, astfel că sistemul pare a avea o memorie mult mai mare decât memoria reală. Cuvântul "realitate" se referă la mediul perceput de om prin intermediul simţurilor. Deoarece "realitatea" este percepută prin intermediul simţurilor, este posibilă "simularea" acesteia prin furnizarea datelor percepute de unul sau mai multe simţuri. De aceea, *Realitatea Virtuală* se referă la modalitatea prin care calculatorul modifică modul în care o persoană percepe realitatea, prin simularea unei alte realităţi. Această realitate, sau mediu, simulată de calculator este numită *Realitate Virtuală*.

Alte definiţii ale termenului *Realitate Virtuală*:
"Un sistem de *Realitate Virtuală* este o interfaţă care implică simularea în timp real şi interacţiunea prin intermediul mai multor canale senzoriale. Aceste canale sunt simţurile omului: văzul, auzul, simţul tactil, mirosul şi gustul" [Burd93].

"*Realitate Virtuală* este un sistem folosit pentru a crea o lume artificială pentru un utilizator astfel încât să aibă impresia că se află în această realitate în care se poate mişca şi interacţiona cu obiectele înconjurătoare" [Man95].

"*Realitatea Virtuală* permite explorarea unei lumi generate de calculator, fiind parte din acea lume artificială" [Sher92].

"*Realitatea Virtuală* este iluzia participanţilor de a participa într-un mediu sintetic, şi nu observarea externă a acelui mediu" [Gig93].

Dintre cele cinci simţuri care sunt folosite pentru percepţia realităţii, nu toate sunt la fel de importante în crearea unui mediu virtual. Simţul gustului şi al mirosului au acţiuni limitate în perceperea realităţii (cu excepţia servirii mesei!) şi puţine cercetări au fost efectuate pentru folosirea lor în medii virtuale.

Simţul tactil este mult mai util, mai ales atunci când se manipulează obiecte în mediul virtual. Greutatea, temperatura, duritatea, rezistenţa la efort, toate aceste informaţii se obţin prin simţul tactil. Din acest motiv, cercetări importante au fost făcute pentru a simula "atingerea" obiectelor virtuale şi, în momentul de faţă, experimentele în mediul virtual permit generarea informaţiilor tactile.

Cele mai importante simţuri folosite în realitatea virtuală sunt văzul şi auzul, deoarece cele mai multe informaţii despre mediul înconjurător se obţin prin intermediul ochilor şi al urechilor. Din acest motiv, cele mai multe cercetări în domeniul realităţii virtuale au fost făcute în domeniul generării imaginii şi a sunetelor în medii virtuale. Chiar clasificarea sistemelor de realitate virtuală se bazează în principal pe modul de generare a imaginii mediului virtual.

Dintre aplicaţiile cele mai importante ale realităţii virtuale se pot enumera:

- Simulatoare de antrenament, în special simulatoare de zbor, în care se pot exersa manevre dificile, fără a pune în pericol viaţa pilotului sau securitatea aparatului de zbor.
- Proiectare în diferite domenii de activitate (construcţii, arhitectură). Proiectantul are posibilitatea să vadă rezultatele proiectului sub forma imaginii acestuia în timp real, să observe detaliile împreună cu alte persoane interesate, şi să ia decizii de modificare înainte de construirea prototipului.
- Vizualizarea ştiinţifică, prin care se obţine imaginea diferitelor modele sau fenomene inaccesibile observaţiei directe (structuri atomice, fluxuri de informaţie, etc).
- În domeniul medical, în special chirurgie, se pot efectua experimente "la rece" de învăţare a diferitelor proceduri, fără riscul vieţii pacientului.
- Jocurile distractive şi filmele de animaţie sunt unele din cele mai cunoscute aplicaţii de realitate virtuală.

1.1 SISTEME DE REALITATE VIRTUALĂ

Se pot distinge mai multe categorii de sisteme de realitate virtuală disponibile în momentul de faţă: sisteme de realitate virtuală imersive (*immersive VR*), sisteme de simulare (*simulation VR*), sisteme proiective (*projected VR*), teleprezenţa (*telepresence*), realitatea îmbogăţită (*augmented reality*) şi sisteme de realitate virtuală desktop (*desktop VR*) [Lou97], [Isd99].

Cea mai completă formă de realitate virtuală se obţine în sistemele imersive. Într-un sistem de realitate virtuală imersiv, contactul participantului cu lumea reală este complet întrerupt, acestuia permiţându-i-se să vadă numai imaginea mediului sintetic, să audă numai sunetele generate artificial şi să interacţioneze numai cu obiectele virtuale pe care le vede în scenă. Acestă incluziune totală a participantului în mediul virtual se obţine prin dispozitive de afişare (display-uri) montate pe cap (*head-mounted display* – HMD), căşti audio (*headphones*), mănuşă de date (data glove) şi îmbrăcăminte de date (*data suits*).

Cea mai cunoscută şi utilizată formă de realitate virtuală este cea din sistemele de simulare (simulatoare). Într-un simulator, participantul este plasat într-o versiune aproape reală a unei cabine de vehicul (avion, elicopter, maşină, tren, navă maritimă, navă spaţială). În această cabină participantul are posibilitatea de a interacţiona cu comenzile de control reale ale vehiculului (manete, pedale, butoane, etc), în timp ce este creată imaginea mediului virtual în care se desfăşoară experimentul, imagine care răspunde acţiunilor efectuate de participant. Acest tip de realitate virtuală, care se mai numeşte şi semi-imersivă, îşi are originea în simulatoarele de zbor construite pentru antrenarea piloţilor.

În sistemele de realitate virtuală proiectivă, imaginea mediului tridimensional este proiectată pe unul sau mai multe ecrane, care pot fi văzute de unul sau mai mulţi utilizatori. Imaginea afişată pe ecrane urmăreşte acţiunile unuia dintre utilizatori, care demonstrează anumite acţiuni sau concepte celorlalţi utilizatori din grup.

În sistemele de teleprezenţă un operator uman este conectat prin intermediul unei interfeţe la senzori de poziţie şi camere video plasate într-un mediu real. În astfel de sisteme, operatorul poate să observe acţiunile unui robot plasat într-un mediu inaccesibil (sau periculos) şi să controleze mişcările acestuia de la o distanţă sigură.

Sistemele de realitate îmbogăţită combină informaţiile generate de calculator cu cele ale unui mediu real [Azu97]. Spre deosebire de sistemele imersive, în care utilizatorul nu are nici un contact cu lumea reală, în realitatea îmbogăţită utilizatorul percepe lumea reală, cu obiecte virtuale suprapuse peste imaginea acesteia. Aplicaţii ale realităţii îmbogăţite se folosesc în medicină, planificarea mişcărilor roboţilor, aviaţia militară, jocuri distractive. În astfel de aplicaţii se foloseşte un dispozitiv de afişare montat pe cap (HMD) care suprapune date generate de calculator peste imaginea mediului real.

În sistemele de realitate virtuală desktop, imaginea vizuală a mediului virtual tridimensional este afişată pe monitorul unui calculator (în general PC).

Participantul interacţionează cu mediul virtual prin dispozitive de intrare standard (tastatură, mouse, joystick). Aceste sisteme permit observarea mediului virtual printr-o "fereastră" (ecranul monitorului) şi de aceea se mai numesc sisteme WoW (*Window On the World*). Sunt cele mai simple şi mai ieftine sisteme de realitate virtuală, dar este de aşteptat ca astfel de sisteme să cunoască în viitor dezvoltări spectaculoase, datorită apariţiei unui mare număr de acceleratoare grafice care permit redarea în timp real a unor imagini realiste.

În realitatea virtuală trebuie să fie captată poziţia corpului uman, astfel încât imaginea redată de display să fie dirijată de mişcările utilizatorului în mediul virtual. De exemplu, întoarcerea capului utilizatorului este sesizată de dispozitivele de captare a poziţiei şi imaginea sau sunetul redat reflectă această nouă poziţie.

Termenul "imersiv" este folosit pentru a descrie un mediu virtual în care participantul (împreună cu mişcările corpului acestuia) fac parte integrantă din mediu. Există diferite grade de imersiune, depinzând de dispozitivele folosite, de hardware-ul şi software-ul disponibil, de gradul de urmărire al mişărilor utilizatorului. Un mediu virtual imersiv are următoarele caracteristici:

- Pentru cel puţin unul din canalele senzoriale importante, tipic cel vizual, toate datele recepţionate de participant sunt cele generate de calculator şi afişate pe display, fiind întreruptă complet recepţionarea datelor corespunzătoare realităţii fizice.
- Sistemul de realitate virtuală conţine modele (geometric, acustic, fizic, tactil), care sunt prelucrate pentru comanda dispozitivelor de redare.
- Imaginile afişate pe display prezintă mediul sintetic într-un mod consistent, cu posibilitatea de percepţie a obiectelor în adâncime, cu mascare reciprocă - vizuală sau auditivă - corespunzătoare.
- În oice moment, datele afişate pe display corespund poziţiei participantului uman în mediul virtual.

Imersivitatea unui sistem de realitate virtuală nu este o caracteristică absolută, şi gradul de imersivitate se poate aprecia într-un mod mai mult sau mai puţin obiectiv. De exemplu, un sistem dotat cu căşti binaurale, date vizuale stereo şi sistem de urmărire a poziţiei capului şi a mâinii este "mai imersiv" decât un alt sistem care nu posedă sisteme de urmărire a poziţiei mâinii sau capului participantului. Imersivitatea este proprietatea sistemului de realitate virtuală care provoacă senzaţia de "prezenţă", adică participantul se simte prezent în "locul" din mediul virtual afişat pe display.

1.2 SCURT ISTORIC AL REALITĂŢII VIRTUALE

Realitatea virtuală, în forma în care este cunoscută în momentul de faţă, nu este o invenţie de dată recentă, experimente care pot fi considerate ca aparţinând acestui domeniu fiind datate cu mai mult de treizeci de ani în urmă.

Dintre acestea, invenţia lui *Morton Heilig* realizată în anul 1962, numită *Sensorama*, este considerată primul sistem de realitate virtuală. Invenţia lui Heilig avea toate caracteristicile unui sistem de realitate virtuală, mai puţin interactivitatea. Observatorul era "plimbat" prin New York pe o motocicletă, având posibilitatea de a simţi vibraţiile motocicletei, curenţii de aer, zgomote şi mirosuri specifice şi, bineînţeles, imaginea străzilor, prezentate ca film color pe ecranul unui televizor montat pe o cască. Ruta de parcurgere era preînregistrată şi, deci, fixă. Heilig nu era inginer, ci specialist în cinematografie, şi dorea să modifice experienţa cinematografică clasică; în orice caz, imaginaţia lui Heilig a devansat timpul în care şi-a desfăşurat activitatea.

Experimentele lui Heilig au fost continuate de un mare specialist în grafica pe calculator, *Ivan Sutherland*, care a adus numeroase contribuţii în acest domeniu. În 1968 Ivan Sutherland a descris un dispozitiv de afişare montat pe cap (*head-mounted display*- HMD), ce consta din două monitoare şi un sistem optic cu oglinzi semitransparente, pe care se afişau imagini dependente de poziţia capului. Acest dispozitiv permitea combinarea unei imagini sintetice stereoscopice cu imaginea reală a mediului, realizând astfel un sistem care în momentul de faţă este numit "realitate îmbogăţită" (*augmented reality*). Sutherland a adus contribuţii importante la generarea imaginilor vizuale prin intermediul calculatorului, proiectând unele dintre cele mai performante generatoare de imagini, realizate de compania Evens&Sutherland.

Un alt nume recunoscut pentru lucrările semnificative în domeniul realităţii virtuale este numele lui *Tom Furness*, care a activat în laboratoarele de cercetări medicale ale forţelor armate americane (*US Air Force's Armstrong Medical Research*). Împreună cu echipa sa, Furness a realizat în anul 1986 un simulator de zbor dotat cu o cabină sofisticată, iar pilotul era echipat cu un sistem de afişare montat pe cap, prin care putea obţine diferite informaţii de zbor şi tactică de luptă, prezentate grafic, suprapus peste imaginea mediului sintetic.

Un alt organism guvernamental din SUA interesat de dezvoltarea simulatoarelor se zbor a fost NASA (*National American Space Agency*), care avea nevoie de simulatoare pentru antrenarea astronauţilor, dat fiind că antrenarea acestora în condiţiile reale ale spaţiului cosmic nu este posibilă. Un proiect semnificativ realizat de inginerii de la NASA este proiectul VIEW (*Virtual Intractive Environment Workstation*), care este un sistem multisenzorial de imersiune, incluzând un dispozitiv de afişare montat pe cap (HMD), căşti auditive tridimensionale, sistem de recunoaştere a vorbirii, sistem de urmărire (*tracking*) a mişcării capului şi mâinii, mănuşă de date (DataGlove). În cadrul acestui proiect a fost experimentat un tip nou de mănuşă de date. Jaron Lanier, unul dintre realizatorii acesteia, a devenit mai târziu directorul companiei VPL, una dintre primele companii care a deţinut un patent pentru mănuşi de date şi a comercializat echipamente pentru sisteme de realitate virtuală.

Cele mai importante contribuţii la dezvoltatea realităţii virtuale au fost prilejuite de realizarea simulatoarelor de antrenament, în special simulatoare de zbor în domeniul militar. Dar, cel puţin până în anii 80-85, majoritatea descoperirilor în domeniul echipamentelor simulatoarelor militare erau secrete şi

nu se publicau. Situaţia s-a schimbat mai târziu, când diminuarea bugetelor militare a determinat direcţionarea către aplicaţii civile ale realităţii virtuale, ca şi posibilitatea de a fi cunoscute şi utilizate echipamente complexe de navigare în mediul virtual.

Dezvoltarea simulatoarelor de zbor a fost aceea care a permis identificarea şi înţelegerea profundă a cerinţelor tehnice care se impun sistemelor de realitate virtuală, impunând ideea că sistemele de realitate virtuală (inclusiv simulatoarele) sunt eficiente numai dacă experienţa este convingătoare din punct de vedere al participantului. Dintre aceste cerinţe de performanţă ale sistemelor de realitate virtuală se pot enumera:

- Viteză de actualizare a datelor afişate de cel puţin 30 cadre/secundă.
- Timp de întârziere trebuie să fie cât mai redus, astfel încât să nu fie perceptibilă diferenţa dintre momentul de execuţie a unei acţiuni şi răspunsul afişat pe display.
- Mediu virtual reprezentat trebuie să fie cât mai complex, cu număr mare de obiecte vizibile.
- Rezoluţia de reprezentare a imaginilor trebuie să fie cât mai mare; includerea mai multor trăsături de realism al imaginilor: ascunderea reciprocă între obiecte, iluminare, umbrire, texturare.
- Simularea mişcării cabinei şi a reacţiei (forţa de răspuns) sistemelor de comandă.

Simulatoarele care prezintă toate aceste caracteristici la nivele acceptabile ating costuri de realizare foarte mari, de câteva milioane de dolari. Pentru simulatoarele de zbor acest cost poate fi acceptabil, dat fiind că economiile de antrenare în simulator a piloţilor, ca şi diminuarea riscului de avariere a aparatelor de zbor în cursul antrenamentelor în condiţii reale, sunt de asemenea considerabile. Pentru ca sisteme de realitate virtuală pentru alte aplicaţii să devină accesibile, trebuie ca performanţe similare celor arătate mai sus să fie obţinute la preţuri mult mai mici. Acest deziderat se conturează totuşi ca un fapt perfect posibil, datorită progreselor tehnologice în domeniile implicate în realitatea virtuală: echipamente de calcul de mare viteză, traductoare de poziţie de mare precizie, display-uri stereografice color, recunoaşterea şi sinteza vorbirii, simularea simţului tactil.

În ţara noastră au fost realizate mai multe simulatoare de antrenament, atât în domeniul civil cât şi în domeniul militar. Primul simulator de antrenament a fost simulatorul de locomotivă Diesel electrică, dezvoltat la Institutul de Cercetări şi Proiectări Tehnologice în Transporturi (ICPTT), în colaborare cu Institutul de Tehnică de Calcul (ITC) în anul 1980. Deşi mijloacele de calcul disponibile în acea perioadă, mai ales în condiţiile din ţara noastră, erau destul de precare, simulatorul a permis antrenarea a numeroşi mecanici de locomotivă, în special pentru rezolvarea diferitelor situaţii de avarie. În condiţii similare au mai fost realizate simulatoare de radar şi de navigaţie. În domeniul simulatoarelor de zbor pentru avioane şi elicoptere militare, primele simulatoare pentru avionul de producţie românească IAR 93 au fost realizate în anii 1984-1989 la INCREST, în colaborare cu ITC, pe baza unor sisteme multiprocesor special proiectate.

În anul 1995, simulatorul elicopterului PUMA, realizat la Institutul de Simulatoare (Simultec), începea, deja, să se apropie de performanţele cerute pentru astfel de sisteme. Calculatorul central este o staţie Silicon Graphics Indigo 2, iar ca generator de imagine s-a folosit un sistem multiprocesor Division, special conceput pentru aplicaţii de realitate virtuală. Cabina echipată asemănător unei cabine reale, sistemul de mişcare cu 6 grade de libertate, simularea eforturilor în comenzi şi a altor sisteme de navigaţie aeriană completează funcţionalitatea unui simulator de zbor cu reale calităţi de antrenament al poliţilor.

În 1998 a fost realizat simulatorul pentru avionul Mig-21 Lancer, la nivelul realizărilor actuale în acest domeniu. Construit pe bază unei staţii Silicon Graphics Onyx2 Infinite Reality, cu o bază de date grafice care acoperă teritoriul întregii ţări, cu sistem de mişcare cu şase grade de libertate, simultorul Mig-21 Lancer este o realizare importantă din ţara moastră în domeniul realităţii virtuale.

1.3 COMPONENTELE SISTEMELOR DE REALITATE VIRTUALĂ

Un sistem de realitate virtuală este compus din mai multe subsisteme care comunică între ele pentru redarea interacţiunii între utilizator şi mediul virtual. În fig. 1.1 este prezentată schema bloc a unui sistem de realitate virtuală.

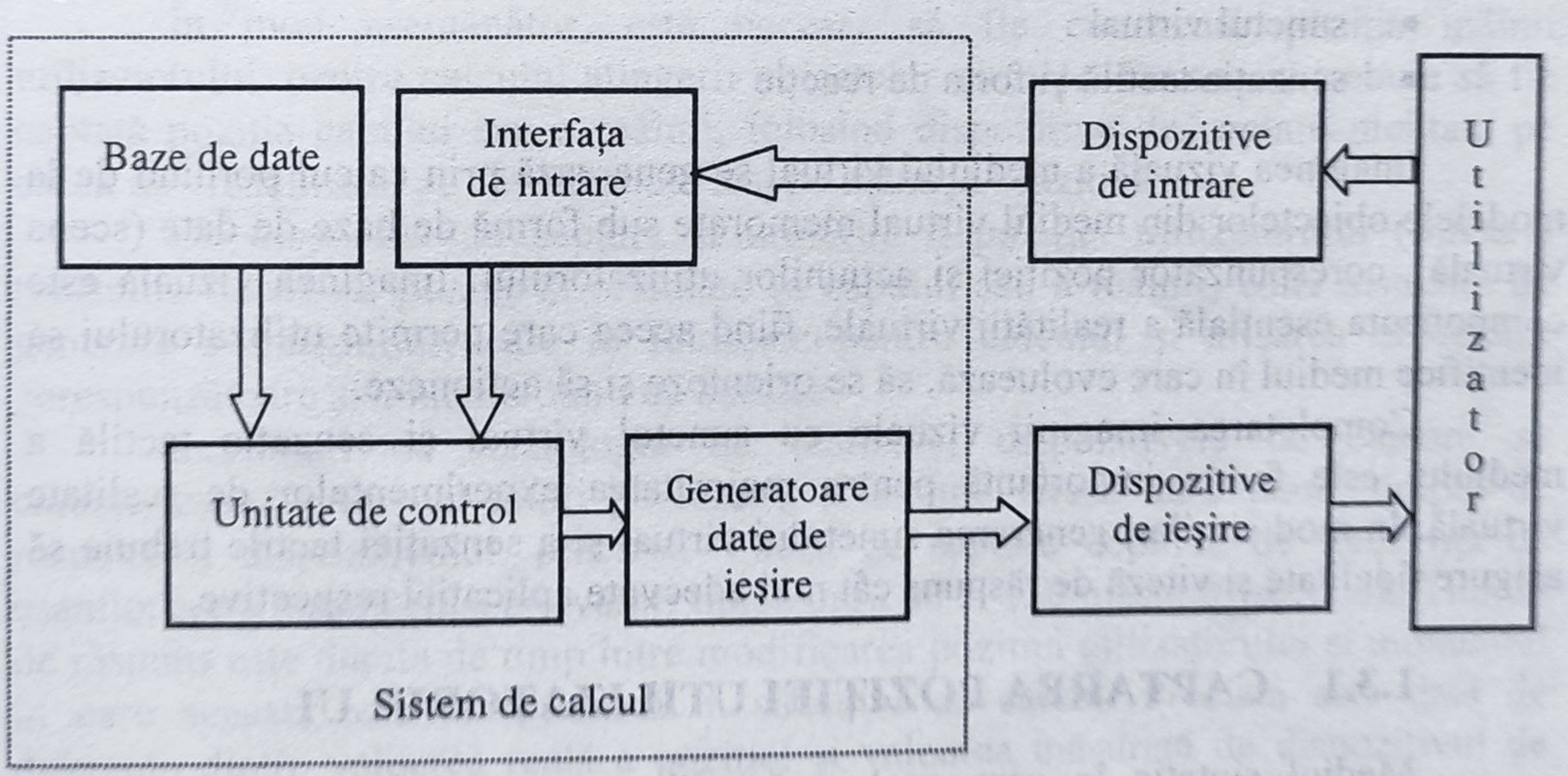


Fig. 1.1 Schema bloc a unui sistem de realitate virtuală.

Un sistem de realitate virtuală este compus dintr-un sistem de calcul, care prelucrează datele de modelare a mediului virtual (conţinute în bazele de date) şi generează imaginile corespunzătoare acţiunilor efectuate de utilizator. Sistemul de calcul implicat în realitatea virtuală este un sistem complex, fiind de cele mai multe ori un sistem paralel sau distribuit, cu interfeţe specializate pentru recepţionarea datelor de la dispozitivele de intrare şi cu generatoare de imagini specializate.

Dispozitivele de intrare specifice realității virtuale sunt traductoare care convertesc acțiunile utilizatorului într-o formă interpretabilă de către calculator. Acțiunile sesizate şi transmise sistemului sunt mişcările (ale capului, ale mâinii, ale corpului) şi vorbirea. Dispozitivele de afişare specifice realității virtuale sunt display-uri video, căşti audio, mănuşi senzitive.

Se observă că un sistem de realitate virtuală este un sistem de control în buclă închisă, unul din elementele buclei fiind utilizatorul, deci este un sistem interactiv. În lucrarea sa [Burd94], G. Burdea defineşte sistemele de realitate virtuală prin trei "I": *imersivitate*, *interactivitate*, la care se adaugă *imaginația* necesară pentru ca mediul virtual creat să imite cât mai fidel realitatea fizică modelată.

Cerințele de imersiune şi interactivitate impun cerința de *sistem de timp real* pentru realitatea virtuală. Pentru ca utilizatorul să se simtă o parte a mediului virtual, trebuie ca sistemul să răspundă actiunilor sale într-un interval de timp suficient de mic, astfel încât diferența dintre momentul de execuție a unei acțiuni şi răspunsul recepționat să nu fie perceptibilă. Această cerință de execuție în timp real impune folosirea unor sisteme de calcul şi dispozitive de intrare şi de ieşire performante, şi de aici provine cea mai mare parte din costurile necesare pentru crearea mediilor virtuale.

Pentru crearea unui mediu virtual se generează şi se redau pe dispozitive corespunzătoare:

- imaginea vizuală
- sunetul virtual
- senzația tactilă şi forța de reacție

Imaginea vizuală a mediului virtual se generează prin calcul pornind de la modelele obiectelor din mediul virtual memorate sub formă de baze de date (scena virtuală), corespunzător poziției şi acțiunilor utilizatorului. Imaginea vizuală este componenta esențială a realității virtuale, fiind aceea care permite utilizatorului să identifice mediul în care evoluează, să se orienteze şi să acționeze.

Completarea imaginii vizuale cu sunetul virtual şi senzația tactilă a mediului este foarte importantă pentru majoritatea experimentelor de realitate virtuală. În mod similar, generarea sunetului virtual şi a senzației tactile trebuie să asigure fidelitate şi viteză de răspuns cât mai adecvate aplicației respective.

1.3.1 CAPTAREA POZIȚIEI UTILIZATORULUI

Mediul sintetic în care evoluează utilizatorul unui sistem de realitate virtuală (numit şi *cybernaut*) este definit într-un sistem de cordonate cartezian tridimensional numit sistem de coordonate universal *(world coordinate` system)*. Fiecare obiect are coordonatele lui relativ la acest sistem de referință şi, de asemenea, utilizatorul are o poziție şi o orientare definită în acest sistem de referință. Toate obiectele pe care utilizatorul le poate vedea la un moment dat se află în interiorul unei piramide numite piramidă de vizualizare, care este o subdiviziune a spațiului determinată de poziția utilizatorului, orientarea capului

acestuia şi unghiul de vizibilitate (aceste noţiuni vor fi descrise pe larg în cap. 3). Obiectele aflate în afara acestui volum se află încă în mediul virtual, dar nu sunt vizibile din poziţia şi orientarea dată a utilizatorului.

Pentru redarea imaginii obiectelor din mediul virtual trebuie, aşadar, să fie captată (urmărită) poziţia şi orientarea capului utilizatorului, care defineşte poziţia şi direcţia de observare, poziţia mâinii utilizatorului, care permite identificarea interacţiunilor cu mediul virtual şi, uneori, poziţia şi a altor părţi ale corpului, pentru redarea mişcării sau a expresiei feţei.

În sistemele grafice convenţionale sau în sistemele de realitate virtuală neimersive se poate urmări poziţia utilizatorului folosind dispozitive ca mouse, trackball sau joystick, cu trei sau şase grade de libertate, disponibile comercial de la firme ca Logitech sau Mouse System Corp. Dispozitivele cu trei grade de libertate permit măsurarea coordonatelor spaţiale x, y, z, iar cele cu şase grade de libertate măsoară în plus rotaţiile după cele trei axe.

În realitatea virtuală imersivă este necesară corelarea comenzilor senzoriale cu poziţia şi acţiunile utilizatorului, pentru a asigura senzaţia de imersiune. Dacă se urmăreşte poziţia capului utilizatorului, atunci când acesta întoarce capul într-o anumită direcţie, imaginea afişată pe display corespunde acelei direcţii, şi senzaţia de imersiune a utilizatorului este foarte puternică. În plus, cunoscând poziţia capului, se poate calcula corect imaginea stereoscopică, compusă din două imagini diferite, corespunzătoare poziţiilor diferite ale celor doi ochi, şi sunetul virtual, corespunzător poziţiei celor două urechi.

În mod asemănător, este necesar să fie cunoscută poziţia mâinii utilizatorului, pentru calculul atingerii obiectelor virtuale. De aceea, trebuie să fie captată poziţia capului sau a mâinii, folosind dispozitive de captare montate pe casca de vizualizare, pe căştile audio sau pe mănuşa senzitivă.

Un dispozitiv de captare şi urmărire a poziţiei utilizatorului (*tracker*) transmite datele de poziţie şi orientare (a capului sau a mâinii) către sistemul de generare a imaginilor, care le foloseşte pentru calculul şi afişarea obiectelor corespunzătoare şi a interacţiunii cu acestea.

Indiferent de tehnologia de realizare, dispozitivele de captare se caracterizează prin mai mulţi parametri, şi anume viteza de captare, timpul de răspuns al dispozitivului, precizia. Viteza de captare depinde de frecvenţa de eşantionare a măsurării şi are valori tipice între 30 şi 120 măsurări/secundă. Timpul de răspuns este durata de timp între modificarea poziţiei utilizatorului şi momentul în care aceasta este recepţionată în sistemul de calcul. Precizia este dată de diferenţa dintre valoarea reală a poziţiei şi valoarea măsurată de dispozitivul de captare; cu cât această diferenţă este mai mică, cu atât precizia este mai bună. Dispozitivele de captare a poziţiei în sistemele de realitate virtuală trebuie să aibă precizie cât mai bună, viteză de captare cât mai mare şi timp de răspuns cât mai mic. Dispozitivele care pot fi folosite pentru captarea poziţiei utilizatorului în spaţiul tridimensional sunt de mai multe tipuri: magnetice, cu ultrasunete, optice.

Dispozitivele de captare magnetice sunt compuse dintr-un ansamblu emiţător-receptor. Emiţătorul este montat într-o poziţie fixă şi generează un câmp magnetic de joasă frecvenţă care este recepţionat de antenele receptorului, ataşat

utilizatorului aflat în mişcare. Din valoarea semnalului recepţionat se poate determina poziţia relativă a receptorului faţă de emiţător. Pentru determinarea poziţiei capului, receptorul se montrează pe casca de vizualizare; pentru determinarea poziţiei mâinii, receptorul se montează pe mănuşa de date. Aceste dispozitive au ca dezavantaje un timp de răspuns ridicat şi perturbarea câmpului magnetic de către obiectele metalice din raza de acţiune a emiţătorului. Companii care produc dispozitive de captare magnetice sunt Polhemus şi Ascension.

Dispozitivele de captare cu ultrasunete folosesc emiţătoare şi receptoare de ultrasunete pentru determinarea poziţiei în spaţiu a utilizatorului. Emiţătorul este compus din trei difuzoare ultrasonore şi este montat într-o poziţie fixă, iar receptorul conţine trei microfoane şi este montat pe casca sau pe mănuşa utilizatorului. La fel ca şi în cazul dispozitivelor magnetice, semnalul captat de receptor permite determinarea poziţiei relative a receptorului faţă de emiţător. Dispozitive de captare cu ultrasunete sunt produse de firmele Logitech şi Transition State.

Dispozitivele de captare optice folosesc grile de diode electroluminescente (*Light Electroluminescent Diodes* – LED) dispuse în poziţie fixă şi o cameră video montată pe casca utilizatorului. Diodele sunt activate în impulsuri, iar semnalul recepţionat de cameră este prelucrat pentru identificare poziţiei relative a camerei faţă de grila de diode. Dezavantajul major al acestor dispozitive îl constitue dimensiunea limitată a grilei de diode, care limitează unghiul de rotaţie ce poate fi măsurat. Una din companiile care produc dispozitive de captare optice este Origin Instruments.

1.3.2 GENERAREA IMAGINII VIZUALE

Generarea imaginii vizuale în realitatea virtuală implică două aspecte importante: crearea modelului scenei virtuale şi vizulizarea scenei virtuale. Crearea modelului scenei virtuale (mai pe scurt, crearea scenei virtuale) este un proces off-line şi, de cele mai multe ori, de durată considerabilă, prin care se creează colecţia de modele ale obiectelor tridimensionale care constitue cea mai adecvată reprezentare a mediului virtual. Vizualizarea scenei virtuale este un proces on-line, care se desfăşoară în timp real, cu participarea uneia sau mai multor persoane, în care scena virtuală este explorată în mod interactiv şi, în fiecare moment, imaginea scenei redată pe display depinde de condiţiile de explorare (poziţie de observare, acţiuni interactive, etc).

Frecvenţa de redare a imaginilor vizuale succesive (*update rate*) trebuie să fie mai mare decât frecvenţa limită a percepţiei vizuale a imaginilor distincte, care se situează în jurul a 15-20 imagini/secundă. Frecvenţa de redare a imaginii depinde de frecvenţa de refresh a display-ului folosit, fiind un submultiplu al acesteia. În mod obişnuit, în realitatea virtuală actualizarea imaginii afişate se face la un multiplu întreg de cadre baleiate. La frecvenţe de refresh a display-ului de 60 cadre/secundă se generează imagini cu frecvenţa de 30 imagini/secundă sau 60 imagini/secundă.

Pentru crearea unei imagini vizuale cât mai apropiate de realitatea fizică a mediului modelat, este necesar ca scena virtuală să conțină cât mai multe obiecte, redate cât mai realist. Generarea imaginii unui număr mare de obiecte ale scenei virtuale într-un interval de timp impus necesită o putere de calcul considerabilă şi de aceea generatoarele de imagine vizuală din sistemele de realitate virtuală sunt realizate pe baza unor arhitecturi paralele specializate care implementează accelerarea hardware a operațiile grafice.

Cel mai important indice de performanță al unui generator de imagine vizuală este *viteza de generare a imaginii*, specificată prin numărul de poligoane redate pe secundă. Acest mod de specificare rezultă din faptul că, în sistemele grafice utilizate în realitatea virtuală, obiectele tridimensionale sunt reprezentate în mod obişnuit printr-o colecție de poligoane.

Calitatea imaginii generate este caracterizată de mai mulți parametri, ca de exemplu rezoluția, modelul de umbrire, capacitatea de texturare, filtrarea anti-aliasing, etc. Rezoluția imaginii se exprimă prin numărul de elemente de imagine (pixeli) afişate pe display. Rezoluțiile la care se generează imaginea în sistemele de realitate virtuală variază de la 640×480 pixeli până la valori de 2048×1280 pixeli. Semnificația acestor parametri poate fi înțeleasă mai bine după parcurgerea capitolelor următoare; în această parte se vor aminti succint aceşti parametri pentru o idee de ansamblu asupra generatoarelor de imagine vizuale disponibile.

Pentru generarea imaginii în sistemele de realitate virtuală sunt necesare mai multe componente hardware (generatoare de imagine, dispozitive de afişare) şi componente software. Aceste componente vor fi descrise în continuare.

1.3.2.1 Generatoare de imagine

Un generator de imagine vizuală poate fi un echipament separat, care se conectează la un calculator gazdă (host), sau poate fi o componentă a sistemului de calcul, numită subsistem grafic sau accelerator grafic. Stațiile grafice folosite în sistemele de realitate virtuală conțin subsisteme grafice puternice pentru generarea în timp real a imaginii obiectelor tridimensionale.

În ultimii ani au fost construite numeroase generatoare de imagine cu performanțe ridicate pentru aplicații de realitate virtuală, la prețuri de la câteva zeci de mii de dolari la câteva sute de mii de dolari. Performanțele şi prețurile acestora au o dinamică extraordinar de mare, astfel încât o prezentare făcută la un moment dat poate deveni depăşită foarte curând. De aceea, vor fi prezentate succint doar câteva generatoare reprezentative, care să dea o idee a performanțelor ce se pot atinge în generarea imaginilor: ProVision (produs de compania Division), Evens&Sutherland, Thomson CSF, stațiile grafice Silicon Graphics.

Generatorul de imagine vizuală ProVision, produs de firma Division în anii 1992-1996, a fost realizat într-o arhitectură paralelă, compus din 10 procesoare de uz general (3 procesoare Intel i860, 2 transputere Inmos T805 şi 5 transputere Inmos 425) pentru calcule geometrice şi o matrice de 64×128 = 8192 de procesoare specializate care efectuează calculele de redare grafică cu umbrire şi texturare [Grim93].

Un astfel de generator de imagine are o performanţă de 30 000 poligoane/secundă cu umbrire Gouraud, la o rezoluţie de maximum 1024×1024 pixeli. Mai multe astfel de generatoare conectate la un calculator gazdă (host), care poate fi o staţie Sun sau un calculator PC, permit generarea imaginii pe mai multe ecrane (canale de imagine), obţinându-se astfel unghiuri de vizibilitate mari.

Generatoarele de imagine Evens&Sutheraland au fost realizate în mai multe variante cu performanţe ridicate: ESIG-2000, ESIG-3000, ESIG-4000. Un generator de imagine ESIG (*Evens-Sutherland Image Generator*) este realizat într-o arhitectură paralelă constând din procesoare specializate (procesoare aritmetice şi procesoare de prelucrare a pixelilor), conectate la un calculator host (fig. 1.2).

Viteza de generare a imaginii a unui generator ESIG 4000 este de 9×10^6 de poligoane/secundă la o rezoluţie de 2048×1280, la care se adaugă numeroase caracteristici de realism a imaginii redate: texturare cu interpolare triliniară, anti-aliasing, Z-buffer, simularea ceţii, etc. Bineînţeles, preţurile sunt destul de ridicate (câteva sute de mii de dolari) şi aceste generatoare sunt utilizate în special în simulatoarele de zbor.

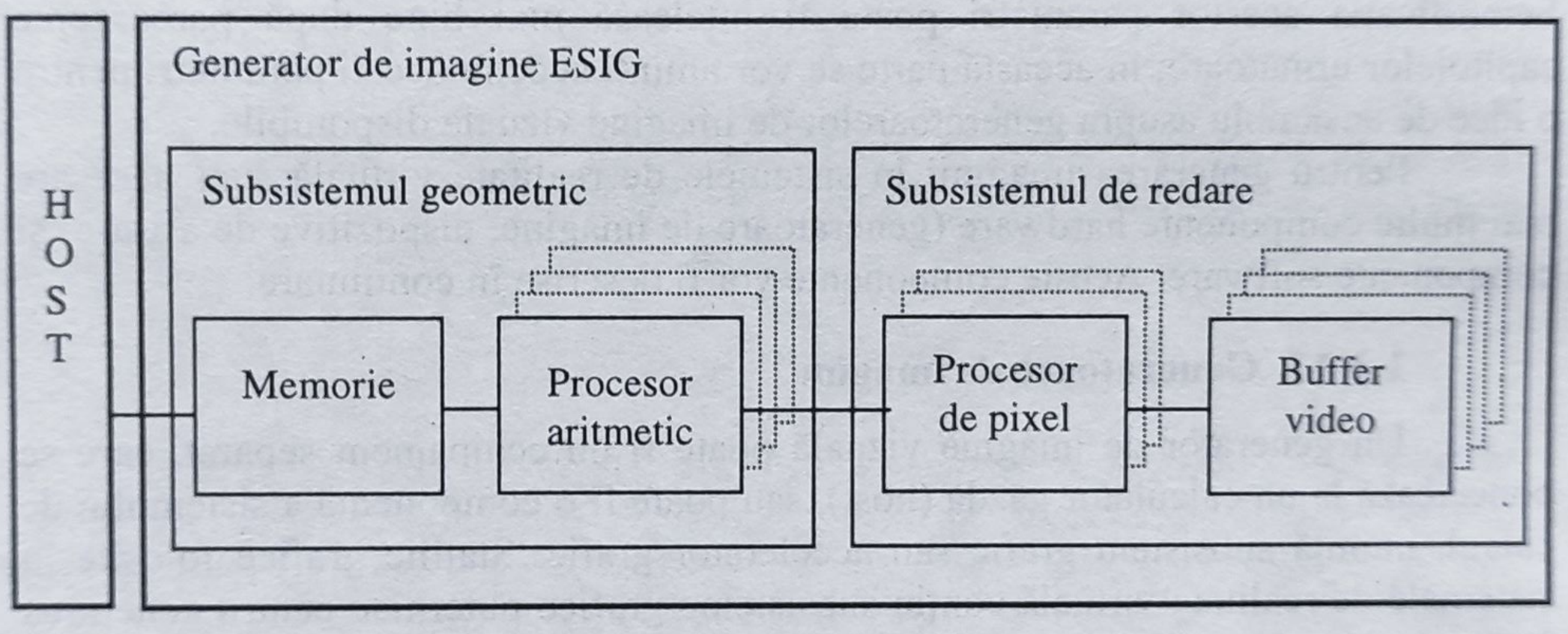


Fig. 1.2 Arhitectura generatoarelor de imagine ESIG.

Din aceeaşi categorie de generatoare de imagine cu performanţe şi preţuri ridicate, folosite în special în simulatoarele de zbor, se mai pot aminti: Real 3D, produs de firma Lockheed Martin; P10 şi Barracuda, produse de firma Primary Image; Ivex Visual System, produs de firma Ivex; Maxvue, produs de firma CAE Electronics; Star G2000, produs de firma Star Technology; Apogée, produs de firma Sogitec; VISA, produs de firma Thomson-CSF.

Staţiile Silicon Graphics. O soluţie frecventă pentru generarea imaginii în realitatea virtuală este folosirea supercalculatoarelor de vizualizare, cum sunt staţiile Silicon Graphics [SGI96]. Combinaţia dintre unitatea centrală în arhitectură multiprocesor puternic şi scalabil şi un subsistem grafic, de asemenea scalabil, permite selectarea unei variante optime ca preţ şi performanţe pentru o aplicaţie dată. În ultimul deceniu au fost produse o gamă întreagă de variante ale staţiilor

Silicon Graphics: Indigo2, Indy, Octane, O2, Onyx2, care au preţuri de la câteva zeci de mii de dolari până la câteva sute de mii de dolari, în funcţie de numărul de procesoare şi subsistemul grafic inclus. Staţia Onyx2 reprezintă una din soluţiile cele mai puternice de realizare a sistemelor de realitate virtuală. Pe lângă capacitatea de redare a imaginii tridimensionale, staţia Onyx2 integrează numeroase alte interfeţe şi dispozitive de creare a mediului virtual: captarea poziţiei utilizatorului, mănuşă de date, dispozitiv HMD, sistem de proiecţie.

Arhitectura staţiei Onyx2 constă dintr-un număr de maximum 12 noduri de procesare, conectate între ele printr-o reţea de interconectare hipercub. Fiecare nod conţine unul sau două procesoare şi 4 GB de memorie, deci sistemul admite maximum 24 de procesoare R10000 (produse de firma MIPS) şi 48 GB de memorie. La fiecare staţie se pot conecta unul sau mai multe (maximum patru) subsisteme grafice pentru prelucrarea şi generararea imaginii cu performanţe ridicate. Subsistemele grafice (ultimele versiuni, InfiniteReality şi RealityMonster) sunt sisteme paralele specializate, care implementează hardware operaţii geometrice şi de redare a primitivelor grafice cu trăsături avansate de realism, cum ar fi texturarea, iluminarea, umbrirea, anti-aliasing. Subsistemul grafic InfiniteReality conţine un pipeline grafic, compus dintr-o succesiune de trei subsisteme (fig. 1.3):

- Subsistemul geometric (*Geometry Engine* – GE)
- Subsistemul de rastru (*Raster Manager* – RM)
- Subsistemul de afişare (*Display Generator* – DG)

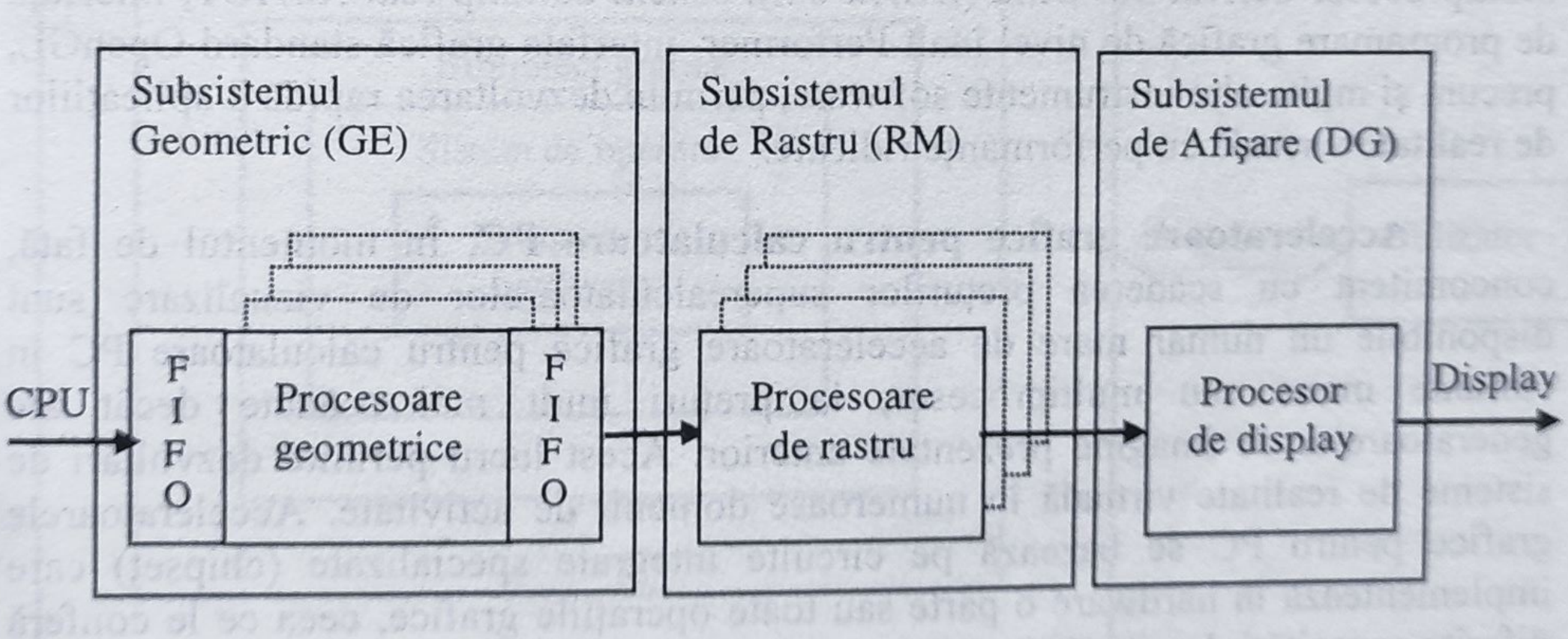


Fig. 1.3 Arhitectura subsistemului grafic InfiniteReality.

Subsistemul geometric execută primul stagiu de procesare grafică pipeline asupra datelor (transformări geometrice, calcule de iluminare, convoluţii, histograme) prin intermediul unor acceleratoare hardware dedicate. Subsistemul geometric este, la rândul lui, un multiprocesor, compus din procesoare specializate numite maşini geometrice (*Geometry Engine* – GE), conectate între ele. Conectarea subsistemului geometric cu celelalte subsisteme este realizată prin intermediul unor buffere FIFO de dimensiuni mari, ceea ce asigură posibilitatea ca acesta să poată

executa continuu, fără întreruperi, indiferent de starea celorlalte segmente ale pipeline-ului.

Subsistemul de rastru constă din una sau mai multe plăci de tip *Raster Manager* (RM), compuse din procesoare grafice specializate (procesoare de rastru) şi buffere de imagine. Un subsistem de rastru poate conţine una, două sau patru plăci RM, care asigură reprezentarea a 2,62 Mpixel (milioane de pixeli), respectiv 5,24 Mpixel şi 10,48 Mpixeli. Modulele RM primesc de la subsistemul geometric date care descriu primitive grafice (puncte, linii, triunghiuri), pe care le transformă în imagine de pixeli în bufferul de imagine (*frame-buffer*). Prin utilizarea extensivă a paralelismului, subsistemul de rastru asigură prelucrarea anti-aliasing, a texturilor şi a efectelor atmosferice, la nivel de pixel în timp real.

Subsistemul de afişare preia imaginile memorate în bufferul de imagine de către subsistemul de rastru şi procesează pixelii prin convertire digital-analogă, pentru a produce fluxul de valori analogice a pixelilor, adecvat afişării pe video-monitoare sau video-proiectoare RGB de înalta rezoluţie. Subsistemul de afişare permite programarea formatelor de afişare, pentru o varietate de monitoare, rezoluţii, frecvenţe de refresh şi caracteristici de întreţesere.

Un pipeline grafic InfiniteReality cu 4 procesoare geometrice GE, o placă de controler de rastru RM şi un subsistem de afişare DG programat pentru o configuraţie cu rezoluţie 1280×1024, 60Hz neîntreţesut, are performantele grafice de vârf de 11×10^6 poligoane/secundă, cu texturare, anti-aliasing, Z-buffer.

Suportul software al staţiilor Silicon Graphics, incluzând sistem de operare multiprocesor derivat din Unix (IRIXx 6.x), sistem de timp real REACT, interfaţa de programare grafică de nivel înalt Performer, interfaţa grafică standard OpenGL, precum şi multe alte instrumente software, permite dezvoltarea rapidă a aplicaţiilor de realitate virtuală cu performanţe ridicate.

Acceleratoare grafice pentru calculatoare PC. În momentul de faţă, concomitent cu scăderea preţurilor supercalculatoarelor de vizualizare sunt disponibile un număr mare de acceleratoare grafice pentru calculatoare PC în variante mono sau multiprocesor, la preţuri mult mai scăzute decât ale generatoarelor de imagine prezentate anterior. Acest lucru permite dezvoltări de sisteme de realitate virtuală în numeroase domenii de activitate. Acceleratoarele grafice pentru PC se bazează pe circuite integrate specializate (chipset) care implementează în hardware o parte sau toate operaţiile grafice, ceea ce le conferă diferite capacităţi de viteză şi realism a imaginii generate. S-au proiectat şi fabricat mai multe tipuri de circuite grafice specializate, pe baza cărora s-au realizat acceleratoare grafice 3D.

În anul 1996 firma Evens&Sutherland, specializată în generatoare de imagine performante, a realizat chipul RealImage, care accelerează hardware toate primitivele interfeţei grafice OpenGL (această interfaţă este prezentată în capitolul 6). Chipurile RealImage sunt folosite în echiparea plăcilor grafice accelerate 3D AccelGalaxy şi E&S Lightning 1200 pentru calculatoare PC sub sistemul de operare Windows NT 4.0 şi au o viteză de generare de aproximativ 3×10^6 triunghiuri/secundă.

Alte chipset-uri folosite în realizarea acceleratoarelor grafice sunt GLINT de la firma 3DLabs, 3Dfx, TNT NVidia 2, şi apar mereu altele noi.

Dispozitivele folosite pentru afişarea imaginii mediului virtual depind de tipul aplicaţiei. În aplicaţii de realitate virtuală desktop sau de teleprezenţă, dispozitivul de afişare poate fi un singur display color, pe care utilizatorul urmăreşte imaginea mediului sintetic. În sistemele de realitate virtuală imersive se folosesc căşti de vizualizare, astfel că utilizatorul vede numai imaginea sintetică, orice contact vizual cu realitatea fizică fiind complet întrerupt. În simulatoare de antrenament, imaginea se proiectează pe mai multe ecrane juxtapuse folosind mai multe proiectoare color.

1.3.2.2 Componente software de generare a imaginii vizuale

În generarea imaginii vizuale în realitatea virtuală, ca şi în majoritatea aplicaţiilor grafice, intervin mai multe componente software, care permit crearea sau redarea scenelor virtuale (fig. 1.4).

Programele de creare sau redare a scenelor virtuale (programe de aplicaţii) se dezvoltă pe baza unor sisteme de dezvoltare (toolkit-uri) sau direct, prin utilizarea unor biblioteci grafice care asigură interfaţa cu echipamentul hardware prin intermediul driverelor sistemului de operare.

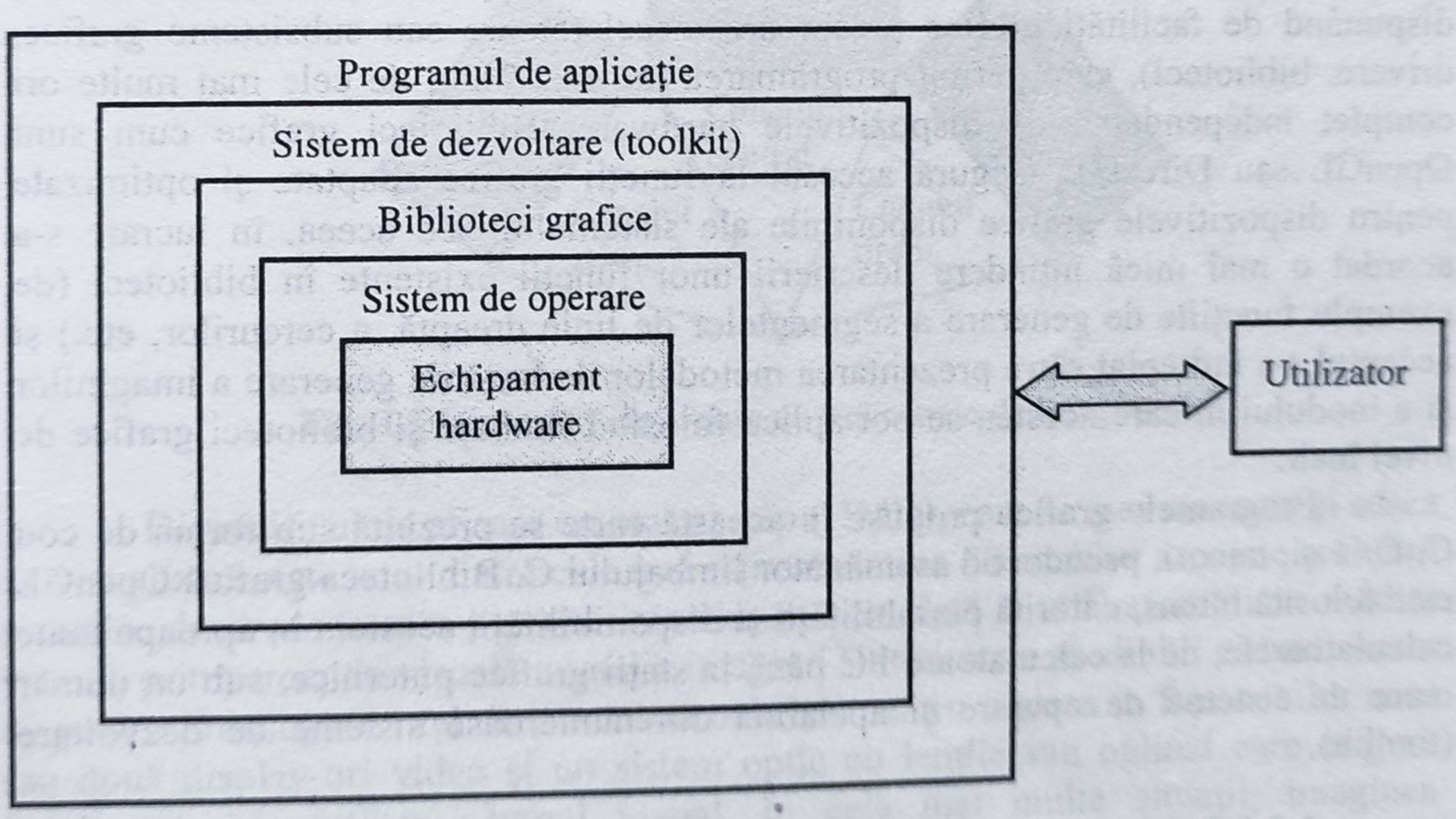


Fig. 1.4 Componentele software de generare a imaginii vizuale.

Sistemele de dezvoltare sunt de cele mai multe ori orientate către aplicaţie şi prevăd un set de funcţii de nivel înalt care permit crearea unui anumit tip de aplicaţie. De exemplu, există toolkit-uri pentru generarea obiectelor şi a scenelor virtuale (3d Studio, Autocad, AC3d, Sense8, Designer Workbench, etc), toolkit-uri pentru redarea imaginii scenelor virtuale (Performer, EasyScene, browser CosmoPlayer, etc).

Bibliotecile grafice sunt pachete de funcţii care asigură interfaţa programului de aplicaţie (creat direct sau prin intermediul unui toolkit care apelează funcţiile bibliotecii) cu echipamentele hardware ale sistemului grafic.

Bibliotecile grafice reprezintă nivelul de programare în care se încearcă introducerea portabilităţii programelor grafice, prin asigurarea unei interfeţe independente de echipamentele hardware care să respecte anumite convenţii de reprezentare a entităţilor grafice descrise în standarde.

Primul standard grafic a fost standardul GKS (*Graphical Kernel System*) elaborat de ISO în 1985, care conţinea un set de funcţii grafice 2D independente de echipament. Acest standard a fost extins în anul 1988 la GKS 3D, care conţine funcţii 3D independente de echipament.

Un alt standard, PHIGS (*Programmer's Hierarchical Graphical System*) este un standard 3D care permite în plus organizarea ierarhică a modelelor obiectelor şi a scenelor virtuale.

Bibliotecile grafice cele mai generale sunt bibliotecile grafice care implementează un anumit standard în definirea funcţiilor de acces la echipamentele hardware. Cele mai cunoscute biblioteci grafice sunt OpenGL, Direct3D, QuickDraw, care sunt implementate în numeroase sisteme grafice.

În momentul de faţă, situaţia cea mai obişnuită în generarea imaginilor în aplicaţiile de realitate virtuală este aceea în care se poate folosi un calculator dispunând de facilităţi grafice (adaptoare, acceleratoare sau subsisteme grafice, drivere, biblioteci), care permit programarea la nivel înalt, de cele mai multe ori complet independentă de dispozitivele hardware. Biblioteci grafice cum sunt OpenGL sau Direct3D asigură accesul la funcţii grafice adaptate şi optimizate pentru dispozitivele grafice disponibile ale sistemului. De aceea, în lucrare s-a acordat o mai mică întindere descrierii unor funcţii existente în biblioteci (de exemplu funcţiile de generare a segmentelor de linie dreaptă, a cercurilor, etc.) şi accentul s-a îndreptat către prezentarea metodelor de bază de generare a imaginilor şi a modului în care acestea se pot aplica folosind limbaje şi biblioteci grafice de nivel înalt.

Programele grafice propuse în această carte se prezintă sub forma de cod C, C++ şi, uneori, pseudocod asemănător limbajului C. Biblioteca grafică OpenGL este folosită intens, datorită portabilităţii şi disponibilităţii acesteia în aproape toate calculatoarele, de la calculatoare PC până la staţii grafice puternice, sub un număr mare de sisteme de operare şi apelabilă din numeroase sisteme de dezvoltare (toolkit).

1.3.2.3 Dispozitive de afişare

În sistemele de realitate virtuală se utilizează trei categorii de *dispozitive de afişare:* display-uri, proiectoare şi dispozitive de afişare montate pe cap (*head mounted display* - HMD).

Display-urile şi proiectoarele sunt folosite în sistemele de realitate virtuală neimersive sau semi-imersive, deoarece utilizatorul plasat în faţa unei imagini afişate pe display sau proiectate pe un ecran poate să mai perceapă şi elemente ale realităţii fizice. Imaginile afişate pe un singur display sau proiectate de un singur

proiector au un câmp de vizualizare redus, tipic 35-45°. Astfel de sisteme de afişare se folosesc în sistemele de realitate virtuală desktop, unde display-ul de afişare este chiar consola calculatorului.

În sistemele de realitate virtuală semi-imersive (cum sunt simulatoarele de zbor), un câmp de vizualizare atât de redus limitează posibilitatea de orientare a utilizatorului în mediul virtual şi de aceea se folosesc mai multe display-uri juxtapuse sau proiectoare care proiectează imagini juxtapuse. Imaginea afişată de fiecare display sau proiector este un canal de imagine, iar imaginea totală a mediului virtual este o imagine multicanal.

Pentru generarea unor imagini multicanal se folosesc sisteme de vizualizare care permit crearea de imagini juxtapuse (acest aspect este explicat în capitolul 4) şi dispozitive de afişare complexe, compuse din mai multe display-uri montate adecvat sau proiectoare şi ecrane corespunzătoare.

În fig. 1.5 este prezentat un dispozitiv de afişare multicanal (cu 3 canale) compus din trei monitore color montate juxtapus. Dispozitivele de afişare multicanal cu monitoare sau proiectoare sunt prevăzute cu posibilități de aliniere şi reglare a imaginilor juxtapuse (*edge blending*).

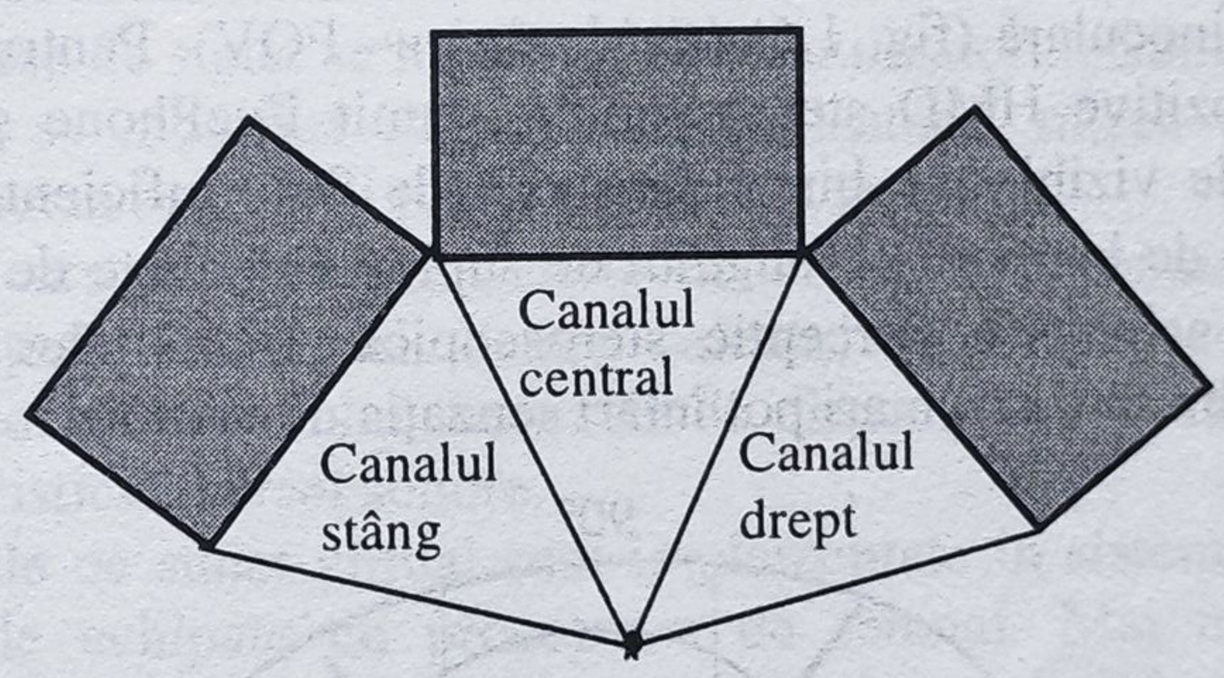


Fig. 1.5 Dispozitiv de afişare cu trei canale de imagine.

Dispozitivul de afişare montat pe cap (HMD), care se mai numeşte şi cască de vizualizare, este unul dintre cele mai reprezentative dispozitive asociate realității virtuale, deoarece acesta permite imersiunea completă a utilizatorului în mediul virtual, prin percepția vizuală numai a imaginii sintetizate a mediului virtual. Acest dispozitiv foloseşte un fel de cască sau ochelari pentru a plasa în fața ochilor unul sau două display-uri video şi un sistem optic cu lentile sau oglinzi care asigură focalizarea imaginii în câmpul vizual. În cele mai multe situații, imaginea percepută de un utilizator echipat cu o cască de vizualizare este imaginea generată pe display-ul căştii, dar există şi căşti de vizualizare cu sistem optic cu oglinzi semitransparente, care permit suprapunerea imaginii generate cu imaginea mediului real. Astfel de căşti sunt folosite în realitatea îmbogățită.

Cele mai multe dintre dispozitivele HMD au două display-uri pentru afişarea imaginilor stereoscopice. Altele au un singur display mai mare pentru redarea cu rezoluție ridicată a imaginii video monoscopice.

Depărtarea obiectelor (adâncimea) poate fi percepută şi în vederea monoculară, dar este mult mai bine percepută în vederea binoculară. În vederea monoculară, adâncimea este percepută prin intermediul acoperirii între obiecte, a umbrelor sau a paralaxei de mişcare, care înseamnă că obiectele mai apropiate par a se deplasa mai mult decât cele mai depărtate.

Cea mai completă percepţie a adâncimii este obţinută în *vederea binoculară (stereoscopică)*, prin care pe retină se creează două imagini ale aceluiaşi obiect, uşor deplasate una faţă de alta. Creierul foloseşte diferenţa dintre poziţia imaginilor create de cei doi ochi pentru a evalua depărtarea obiectului.

În sistemele de realitate virtuală, vederea stereoscopică este obţinută prin plasarea a două display-uri, câte unul în faţa fiecărui ochi. Imaginea generată pe fiecare display este calculată dintr-un punct de observare corespunzător ochiului respectiv. În acelaşi mod ca în vederea normală binoculară, creierul combină cele două imagini pentru a obţine informaţia de depărtare a obiectelor. Un astfel de dispozitiv, numit dispozitiv de afişare stereo montat pe cap, mai necesită şi un dispozitiv de captare a poziţiei care să furnizeze poziţia capului utilizatorului în mediu.

Una din caracteristicile importante ale vederii stereoscopice este unghiul de vizibilitate binoculară (fig. 1.6) (*field of view*- FOV). Pentru unul din cele mai cunoscute dispozitive HMD stereoscopice, numit EyePhone şi produs de firma VPL, unghiul de vizibilitate binoculară este de 90°, suficient de mare pentru a asigura senzaţia de imersie, iar unghiul de suprapunere este de 60,6°, suficient de mare pentru a asigura o percepţie steroscopică. Bineînţeles că dispozitive cu unghiuri de vizibilitate mai mari pot întări senzaţia de imersie.

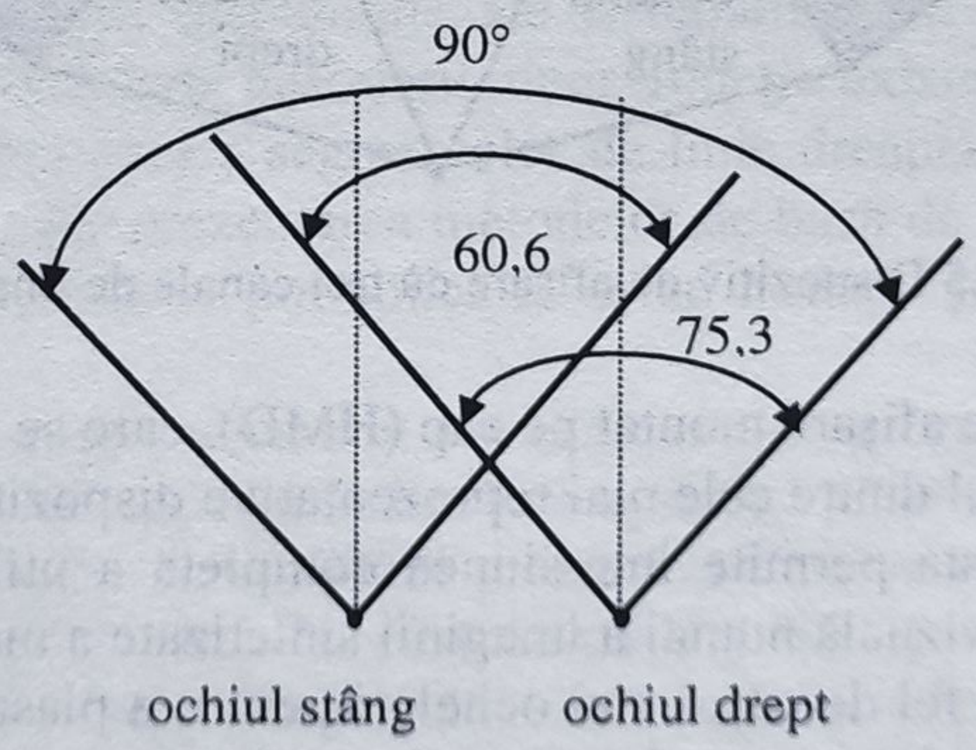


Fig. 1.6 Unghiul de vizibilitate binoculară (90°), de suprapunere (60,6°) şi pentru un ochi (75,3°) a dispozitivului HMD EyePhone (VPL).

Dispozitivul *EyePhone* a fost unul din primele modele de cască de vizualizare prezente pe piaţă. Acesta era echipat cu display-uri Sony cu cristale lichide (*Liquid Crystal Display* - LCD) cu rezoluţia de 360×240 pixeli şi avea un preţ destul de ridicat (aproximativ 11 000 de dolari). În momentul de faţă, dispozitive HMD cu performanţe similare se pot găsi la preţuri mai scăzute, sub

1000 de dolari. Exemple de astfel de căşti de vizualizare sunt Flight Helmet (produsă de Virtual Research Inc), sau Cyberface II (produsă de Leap Systems).

Dispozitivele HMD care folosesc display-uri LCD au o rezoluţie scăzută, ceea ce face ca imaginea afişată să fie de slabă calitate. Soluţia pentru obţinerea unei rezoluţii mai mari este de a se folosi display-uri cu tub catodic (*Cathode Ray Tube*-CRT) în locul celor cu cristale lichide. Astfel de dispozitive echipate cu display-uri CRT sunt disponibile de mai multă vreme pe piaţă, de exemplu HMD-131, produs de compania Virtual Reality Inc., monocrom, cu rezoluţie de 1280×1024 pixeli, sau Datavision 9c, produs de firma n-Vision, cu display-uri color şi rezoluţie de 1280×960 pixeli. Preţul unor astfel de dispozive este însă destul de ridicat (în jur de 70 000 dolari).

1.3.3 GENERAREA SUNETULUI

Sistemele de realitate virtuală cu un grad puternic de imersivitate posedă dispozitive de generare a sunetului virtual tridimensional. Sunetul virtual nu trebuie să fie confundat cu sunetul stereo. Sunetul stereo, generat de două căşti de sunet fixate pe urechi, nu se modifică atunci când utilizatorul mişcă (roteşte) capul, ceea ce dă senzaţia că sursa de sunet se mişcă odată cu utilizatorul. Sunetul virtual este astfel generat încât se modifică atunci când utilizatorul mişcă capul, ceea ce creează senzaţia că sursa sonoră îşi păstrează poziţia în spaţiu.

Căştile prin care se generează sunetul virtual sunt *căşti binaurale* prevăzute cu dispozitiv de captare a poziţiei. Pentru calculul semnalului sonor virtual se folosesc datele de poziţie a capului, furnizate de dispozitivul de captare, şi datele de poziţie şi caracteristicile sursei sonore.

Generatoarele de sunet virtual sunt implementate în sistemele de realitate virtuală prin module echipate cu procesoare de semnal. Mai multe tipuri de generatoare de sunet virtual au fost realizate de compania Crystal River Engineering, la preţuri care variază de la 1000 la 60 000 de dolari. Modelele mai puternice de generatoare de sunet (de exemplu, modelul Acoustetron) permit redarea sunetelor unui număr mare de surse sonore (mai mult de 16 surse sonore).

1.3.4 SIMULAREA SENZAŢIEI TACTILE ŞI A FORŢEI DE REACŢIE

Multe din sistemele de realitate virtuală comerciale se bazează pe imaginea vizuală a mediului înconjurător, neglijând aspectul fizic al realităţii: identificarea tactilă a obiectelor şi reacţia acestora la atingere. Există însă aplicaţii ale realităţii virtuale în care simularea contactului cu obiectele mediului virtual este importantă şi nu mai poate fi neglijată, un exemplu tipic fiind simulatoarele chirurgicale. Este nedorit şi riscant ca un medic să fie antrenat pentru execuţia unei operaţii chirurgicale, fără să aibă senzaţia atingerii organului investigat. În astfel de aplicaţii este necesară creşterea gradului de realism al mediului sintetic prin simularea caracteristicilor fizice ale obiectelor: greutatea, rigiditatea, compoziţia suprafeţei, ş.a.m.d.

Din punct de vedere al terminologiei, în [Burd97] se precizează doi termeni care descriu două senzaţii diferite care intervin la atingerea obiectelor: senzaţia tactilă (*touch feedback*) şi forţa de reacţie (*force feedback*). Senzaţia tactilă este senzaţia recepţionată de piele atunci când este atinsă mecanic, termic, chimic sau electric. Senzorii tactili sunt plasaţi la suprafaţa pielii, cea mai mare densitate a acestora găsindu-se în mână şi furnizează informaţii despre rugozitatea suprafeţelor sau temperatură. În realitatea virtuală se simulează senzaţia tactilă prin acţionarea cu o forţă repartizată spaţial asupra extremităţii degetelor.

Forţa de reacţie este răspunsul unui obiect la o acţiune externă, de exemplu rezistenţa la apăsare sau greutatea pe care o prezintă dacă este ridicat. Forţa de reacţie este recepţionată de senzori plasaţi în interiorul corpului, în general în tendoanele muschiulare. În realitatea virtuală, forţa de reacţie se generează prin dispozitive care produc o forţă de apăsare asupra organismului. În cazul obiectelor virtuale atinse cu mâna, se generează o forţă de apăsare asupra degetelor corespunzătoare reacţiei obiectelor atinse.

În afară de termenii descrişi, se mai folosesc termenii de reacţie haptică (*haptic feedback*, de la termenul grecesc *haptesthai*, sinonom cu simţul tactil) sau reacţie chinestezică *(kinesthesic feedback*).

Interacţiunea utilizatorului cu obiectele mediului virtual este asigurată prin intermediul unui dispozitiv numit mănuşă de date (*data glove*). O mănuşă de date (sau mănuşă senzitivă) este prevăzută cu senzori ai mişcării şi poziţiei mâinii şi cu dispozitive pneumatice pentru generarea forţei de reacţie la atingerea obiectelor.

Diferite modele de mănuşi de date sunt deja disponibile, de exemplu, Rutgers Portable Master, realizată la Universitatea Rutgers [Burd92], LRP Portable Master, realizată la Laboratorul de Robotică din Paris (Laboratoire de Robotique de Paris - LRP), sau Force ArmMaster, realizată de compania EXOS Inc.

1.4 SISTEME DE REFERINŢA TRIDIMENSIONALE

Pentru crearea şi redarea scenelor virtuale este necesar ca obiectele să fie poziţionate într-un sistem de referinţă tridimensional. Există mai multe posibilităţi de a specifica poziţia unei mulţimi de puncte (vârfuri) prin care este reprezentat un obiect în spaţiul tridimensional: coordonate cilindrice, coordonate sferice, coordonate carteziene. Dintre aceste sisteme de referinţă, cel mai utilizat în aplicaţiile grafice este sistemul de coordonate cartezian.

Sistemul de coordonate cartezian în care sunt definite toate obiectele scenei virtuale se numeşte sistem de referinţă universal (*world coordinate system*- WCS).

Un sistem de coordonate cartezian se defineşte prin originea O şi trei axe perpendiculare, Ox, Oy şi Oz, orientate după regula mâinii drepte sau după regula mâinii stângi. Diferenţa dintre cele două sisteme se poate urmări în fig. 1.7.

Într-un sistem orientat după regula mâinii drepte, dacă se roteşte mâna dreaptă în jurul axei z de la axa x pozitivă spre axa y pozitivă, orientarea degetului

mare este în direcţia z pozitiv. Într-un sistem orientat după regula mâinii stângi, rotirea de la axa x pozitivă spre axa y pozitivă, cu orientarea degetului mare în direcţia z pozitiv, se obţine folosind mâna stângă. Orientarea după regula mâinii drepte a sistemelor de coordonate corespunde convenţiei matematice standard.

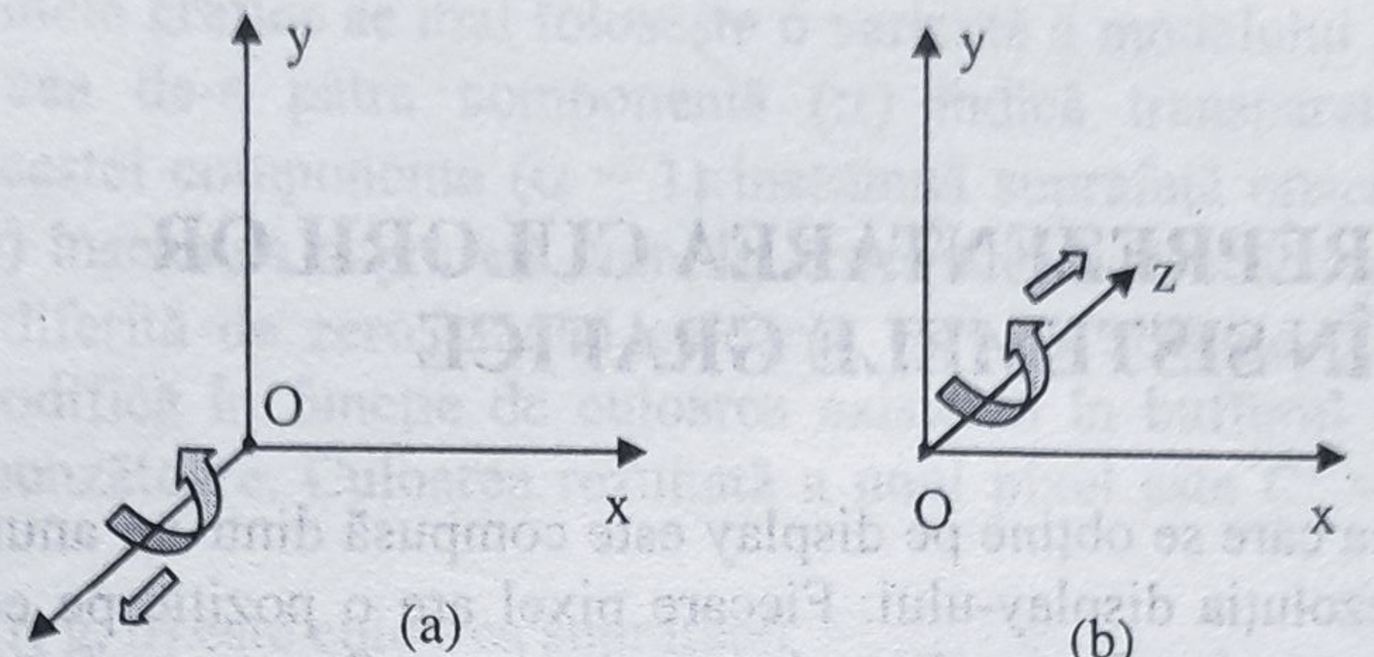


Fig.1.7 (a) Sistem de coordonate orientat după regula mâinii drepte şi (b) după regula mâinii stângi.

Într-un sistem orientat după regula mâinii drepte, dacă se roteşte mâna dreaptă în jurul axei z de la axa x pozitivă spre axa y pozitivă, orientarea degetului mare este în direcţia z pozitiv. Într-un sistem orientat după regula mâinii stângi, rotirea de la axa x pozitivă spre axa y pozitivă, cu orientarea degetului mare în direcţia z pozitiv, se obţine folosind mâna stângă. Orientarea după regula mâinii drepte a sistemelor de coordonate corespunde convenţiei matematice standard.

Diferite sisteme de grafică tridimensională sau de realitate virtuală folosesc convenţii diferite pentru definirea sistemelor de referinţă, ceea ce conduce la confuzii, dacă nu se precizează convenţia folosită. În acest text, pentru sistemul de referinţă universal se foloseşte convenţia de sistem de coordonate drept. În grafica tridimensională se mai folosesc şi alte sisteme de referinţă, care permit descrierea operaţiilor de transformări geometrice şi care vor fi precizate pe parcursul lucrării.

Un punct P în spaţiul tridimensional se reprezintă în sistemul de referinţă cartezian printr-un tripet de valori scalare x, y, z, care reprezintă componentele vectorului de poziţie **OP** pe cele trei axe de coordonate. Dacă se notează cu **i**, **j**, **k** versorii (vectorii unitate) ai celor trei axe de coordonate x, y, z, atunci vectorul de poziţie al punctului P este $\mathbf{OP} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$. În notaţia matriceală, un punct în spaţiul tridimensional se poate reprezenta printr-o matrice linie sau coloană:

$$\mathbf{P} = [x \quad y \quad z] \text{ sau } \mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Ambele convenţii sunt folosite în egală măsură în sistemele grafice, ceea ce, din nou, poate provoca diferite confuzii, dacă nu se precizează convenţia folosită. Convenţia de reprezentare sub formă de matrice linie a unui punct, folosită în unele lucrări [Watt95], [Mold96], are avantajul că exprimă operaţiile de

concatenare a matricelor într-un mod natural, de la stânga la dreapta. Convenția de reprezentare matematică, standardul grafic PHIGS, biblioteca grafică OpenGL, ca şi unele din lucrările de referință în domeniu [Foley90], folosesc notația de matrice coloană pentru un punct în spațiul tridimensional, care este adoptată şi în lucrarea prezentă.

1.5 REPREZENTAREA CULORILOR ÎN SISTEMELE GRAFICE

Imaginea care se obține pe display este compusă dintr-un anumit număr de pixeli, dat de rezoluția display-ului. Fiecare pixel are o poziție pe ecran, dată de adresa lui în ferestra de afişare, şi o culoare care poate fi reprezentată în mai multe modele: modelul RGB, modelul HSV, modelul HLS, şi altele. Dintre aceste modele, în grafică se foloseşte cel mai frecvent modelul RGB.

1.5.1 MODELUL RGB

În modelul RGB, culoarea este reprezentată printr-un triplet de culori primare, roşu (*red*) verde (*green*), albastru (*blue*). Utilizarea preponderentă a modelului RGB în grafică se datorează în primul rând faptului că monitoarele color folosesc acest model de producere a culorii. Culoarea este produsă pe ecranul monitorului prin excitarea a trei puncte adiacente de substanță fosforescentă de culori roşu, verde şi, respectiv, albastru. Punctele fosforescente fiind apropiate, ochiul percepe tripletul de puncte ca un singur punct a cărui culoare este suma celor trei componente. Spațiul de reprezentare a tuturor culorilor în modelul RGB este un cub într-un sistem de coordonate cu axele notate Roşu (*Red*), Verde (*Green*), Albastru (*Blue*) (fig. 1.8).

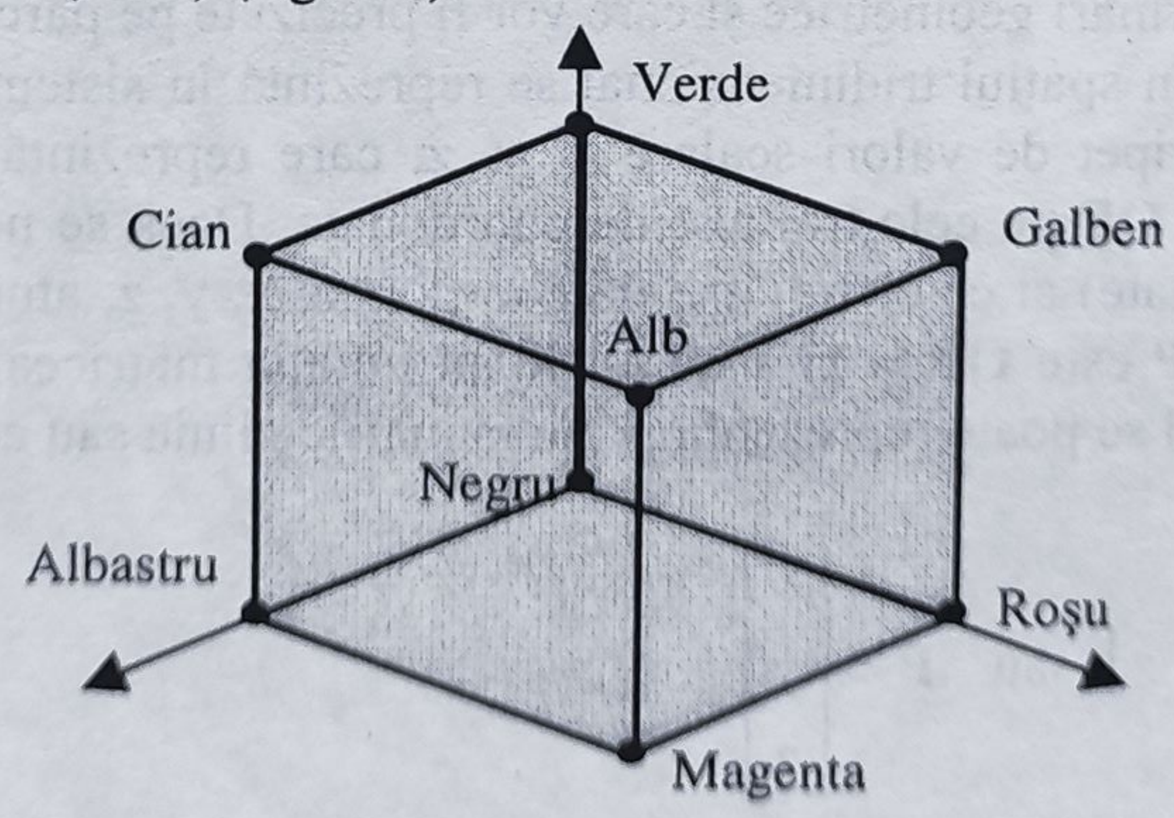


Fig. 1.8 Spațiul culorilor în modelul RGB.

Orice culoare în modelul RGB se exprimă printr-un triplet (r,g,b) şi îi corespunde un punct în spațiul RGB al cărui vector **C** este:

$$\mathbf{C} = r\mathbf{R} + g\mathbf{G} + b\mathbf{G} \tag{1.1}$$

unde **R**, **G**, **B** sunt versorii axelor roşu (*red*), verde (*green*), albastru (*blue*). În acest model culoarea negru este reprezentată prin tripletul (0,0,0), iar culoarea alb este reprezentată prin tripletul (1,1,1).

În sistemele grafice se mai foloseşte o variantă a modelului RGB, modelul RGBA, unde cea de-a patra componentă (α) indică transparența suprafeței. Valoarea 1 a acestei componente ($\alpha = 1$) înseamnă suprafață opacă, iar valoarea minimă ($\alpha = 0$) înseamnă suprafață complet transparentă. Dacă transparența unei suprafețe este diferită de zero, atunci culoarea care se atribuie pixelilor acestei suprafețe se modifică în funcție de culoarea existentă în bufferul de imagine în pozițiile corespunzătoare. Culoarea rezultată a unui pixel este $C' = f(\alpha, C, C_P)$, unde:

- C(r, g, b) este culoarea suprafeței
- α este transparența suprafeței, cu valori între 0 şi 1
- C_P (r_P, g_P, g_P) este culoarea precedentă a pixelului, aflată în bufferul de imagine
- C'(r', g', b') este noua culoare a pixelului, rezultată prin aplicarea unei funcții *f* fiecărei componente de culoare.

Bibliotecile grafice permit definirea mai multor funcții *f* de combinare liniară între culoarea transparentă cu culoarea de fond.

1.5.2 MODELUL HSV

Modelul HSV reprezintă o transformare neliniară a spațiului RGB, prin care culorile sunt specificate prin componentele: nuanță (*hue*), saturație (*saturation*) şi valoare (*value*). Modelul HSV este folosit în definirea mai intuitivă a culorilor (perceptuală), decât definirea prin culori primare. De exemplu, sarcina de a modifica o culoare astfel încât să fie "mai luminosă", "mai galbenă", etc, se realizează mult mai simplu în modelul HSV decât în modelul RGB.

Spațiul culorilor în modelul HSV este o piramidă hexagonală într-un sistem de coordonate polare HVS (fig. 1.9). Definițiile parametrilor H, S, V sunt legate de modul în care este percepută culoarea:

- Nuanța este acea calitate prin care ochiul distinge o familie de culori de altă familie de culori, de exmplu roşu de galben, verde de albastru.
- Saturația, numită şi cromicitate (*chroma*), este calitatea culorii prin care se distinge o culoare puternică de o culoare slabă. Se mai poate defini ca intensitatea culorii sau distanța față de nuanța gri.
- Valoare este calitatea prin care se distinge o culoare luminoasă de o culoare întunecată.

În modelul HVS, variația parametrului H (între 0 şi 360°) corespunde selecției culorii. Scăderea parametrului S (desaturarea culorii) înseamnă adăugare de alb. Scăderea parametrului V (devaluarea culorii) înseamnă adăugare de negru.

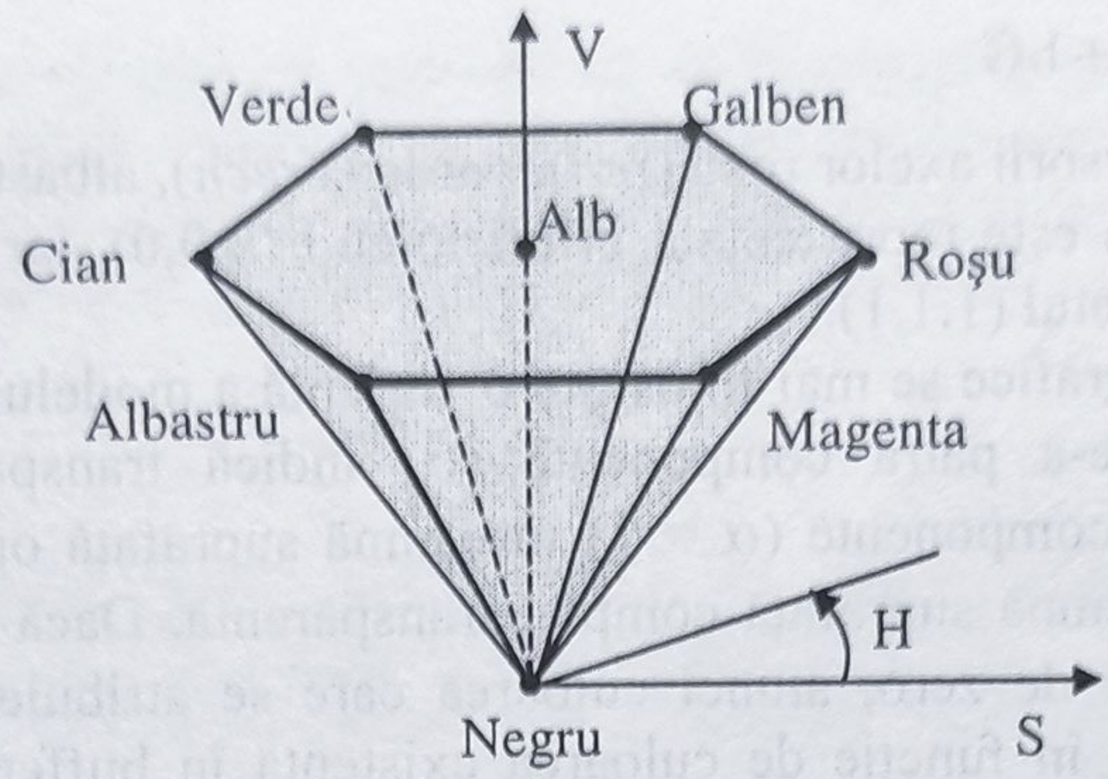


Fig.1.9 Spațiul culorilor în modelul HSV.

Conversia din spațiul RGB în spațiul HSV se înțelege mai uşor într-o interpretare geometrică: hexagonul din reprezentarea HVS se obține prin proiectarea cubului RGB pe un plan normal la diagonala principală, pe direcția diagonalei principale. Corespondența dintre vârfurile hexagonului de bază HSV şi vârfurile cubului RGB este:

RGB		HSV
100	roşu	0,1,1 (H este măsurat în grade)
110	galben	60,1,1
010	verde	120,1,1
011	cian	180,1,1
001	albastru	240,1,1
101	magenta	300,1,1

Pentru aceeaşi definiție a unei culori (de exemplu, prin tripletul rgb) este posibil ca monitoare diferite să genereze culori diferite, datorită caracteristicilor constructive diferite. În general, între componenta de culoare transmisă unui monitor şi intensitatea produsă de ecranul acestuia există relația neliniară:

$$R_m = K(R_i')^{\chi_r};\ G_m = K(G_i')^{\chi_g};\ B_m = K(B_i')^{\chi_b} \qquad (1.2)$$

unde χ ia valori între 2.3 şi 2.8.

Liniarizarea relației între valorile componentelor culorii produse prin program şi intensitățile generate pe ecran se numeşte corecție gamma (*gamma corection*). Corecția gamma se realizează aplicând o transformare care, compusă cu transformarea (1.2), să asigure o relație liniară între culoarea (R_i, G_i, B_i) generată prin program şi culoarea (R_m , G_m, B_m) generată pe ecranul monitorului.

Acestă transformare este:

$$R_i' = k(R_i)^{1/\chi_r};\ G_i' = k(G_i)^{1/\chi_g};\ B_i' = k(B_i)^{1/\chi_b} \qquad (1.3)$$

Corecția gamma se poate implementa printr-o tabelă de căutare inclusă în controlerul grafic, care transformă valorile R_i, G_i, B_i citite din bufferul de imagine în valorile corespunzătoare R_i', G_i', B_i' transmise monitorului. Lipsa corecției gamma, sau o corecție gamma calculată, eronat produce imagini incorecte.

2

MODELAREA OBIECTELOR

Modul cel mai convenabil de creare a scenelor virtuale este acela în care fiecare obiect este modelat într-un sistem de coordonate propriu, numit sistem de referință model (sau sistem de referință local), în care punctele (vârfurile) obiectului sunt precizate relativ la un anumit punct de referință local. De fapt, în modelarea ierarhică, un obiect complex poate avea un număr oarecare de sisteme de referință locale, câte unul pentru fiecare parte componentă a sa. Instanțierea unui obiect în scena virtuală înseamnă amplasarea acestuia în sistemul de referință universal printr-o succesiune de scalări, rotații şi translații, care transformă obiectul din sistemul de referință local în sistemul de referință universal. Această succesiune de transformări este cunoscută sub numele de transformare de modelare.

Proprietățile obiectelor tridimensionale care se modelează în aplicațiile grafice se pot împărți în două categorii: *forma* şi *atribute de aspect*. Informația de formă a unui obiect este diferită de celelalte atribute ale obiectului, deoarece forma este aceea care determină modul în care obiectul apare în redarea grafică şi toate celelalte atribute se corelează cu forma obiectului (de exemplu, culoarea se specifică pentru fiecare element de suprafață a obiectului).

Din punct de vedere al formei, obiectele tridimensionale reprezentate în grafica pe calculator pot fi obiecte *solide* sau obiecte *deformabile*. Un solid este un obiect tridimensional a cărui formă şi dimensiuni nu se modifică în funcție de timp sau de poziția în scenă (proprietatea de formă volumetrică invariantă). Majoritatea aplicațiilor de realitate virtuală se bazează pe scene compuse din solide, dar există şi aplicații în care obiectele reprezentate îşi modifică forma şi dimensiunile într-un mod predefinit sau ca urmare a unor acțiuni interactive (de exemplu, în simulări ale intervențiilor chirurgicale). Chiar şi reprezentarea unor astfel de obiecte (obiecte deformabile) se bazează pe un model al unui solid care se modifică în cursul experimentului de realitate virtuală. În lucrarea de față se vor prezenta modele ale solidelor care stau la baza prelucrărilor din grafică şi realitate virtuală.

Modelarea solidelor este o tehnică de proiectare, vizualizare şi analiză a modului în care obiectele reale se reprezintă în calculator. În ordinea importanței şi a frecvenței de utilizare, metodele de modelare şi reprezentare a obiectelor sunt următoarele:

1. *Modelarea poligonală.* În acestă formă de reprezentare, obiectele sunt aproximate printr-o rețea de fețe care sunt poligoane planare.
2. *Modelarea prin rețele de petice parametrice bicubice* (*bicubic parametric patches*). Obiectele sunt aproximate prin rețele de elemente spațiale numite petice (patches). Acestea sunt reprezentate prin polinoame cu două variabile parametrice, în mod obişnuit cubice.
3. *Modelarea prin compunerea obiectelor* (*Constructive Solid Geometry - CSG*). Obiectele sunt reprezentate prin colecții de obiecte elementare, cum sunt cilindri, sfere, poliedre.
4. *Modelarea prin divizare spațială.* Obiectele sunt încorporate în spațiu, prin atribuirea unei etichete fiecărui element spațial, în funcție de obiectul care ocupă elementul respectiv.

Aceste metode de modelare şi reprezentare a solidelor se pot grupa în reprezentări prin suprafață de frontieră (primele două metode) şi reprezentări prin volum (ultimele două metode).

2.1 MODELAREA POLIGONALĂ A OBIECTELOR

Modelarea poligonală, în care un obiect constă dintr-o rețea de poligoane planare care aproximează suprafața de frontieră (*boundary representation – B-rep*), este forma "clasică" folosită în grafica pe calculator. Motivele utilizării extinse a acestei forme de reprezentare sunt uşurința de modelare şi posibilitatea de redare rapidă a imaginii obiectelor. Pentru obiectele reprezentate poligonal s-au dezvoltat algoritmi de redare eficienți, care asigură calculul umbririi, eliminarea suprafețelor ascunse, texturare, anti-aliasing, frecvent implementați hardware în sistemele grafice. În reprezentarea poligonală, un obiect tridimensional este compus dintr-o colecție de fețe, fiecare față fiind o suprafață plană reprezentată printr-un poligon.

2.1.1 REPREZENTAREA POLIGOANELOR

Un poligon este o regiune din plan mărginită de o colecție finită de segmente de dreaptă care formează un circuit închis simplu.

Fie n puncte în plan, notate v_0, v_1, ...,v_{n-1} şi n segmente de dreaptă $e_0 = v_0v_1$, $e_1 = v_1v_2$, ... $e_{n-1} = v_{n-1}v_0$, care conectează perechi de puncte succesive în ordine ciclică, deci inclusiv conexiunea între ultimul punct şi primul punct din listă. Aceste segmente mărginesc un poligon, dacă şi numai dacă:

(a) Intersecția fiecărei perechi de segmente adiacente în ordinea ciclică este un singur punct, conținut de ambele segemente: $e_i \cap e_{i+1} = v_{i+1}$, pentru oricare $i = 0,\ldots, n-1$.

(b) Segmente neadiacente nu se intersectează: $e_i \cap e_j = \varnothing$, pentru orice $j \neq i+1$.

Segmentele care mărginesc un poligon (linia poligonală) formează un circuit închis (*ciclu*), deoarece segmentele sunt conectate capăt la capăt şi ultimul segment conectează ultimul punct cu primul punct; ciclul este simplu deoarece segmentele neadiacente nu se intersectează.

Punctele v_i se numesc vârfurile poligonului (*vertices*); segmentele e_i se numesc muchii (sau laturi) ale poligonului. De remarcat că un poligon conţine n vârfuri şi n muchii şi că muchiile sunt orientate, astfel încât formează un ciclu (circuit închis). O astfel de orientare a segmentelor se numeşte orientare consistentă. În general, se foloseşte ordinea de parcurgere în sensul invers acelor de ceasornic: dacă se parcurg muchiile în sensul lor de definiţie, interiorul poligonului este văzut întotdeauna în partea stângă (fig. 2.1).

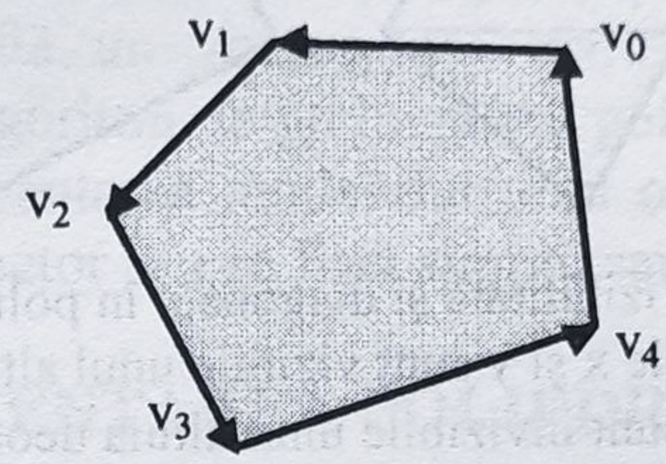


Fig. 2.1 Segmentele liniei poligonale sunt orientate şi nu se autointersectează.

O teoremă importantă în prelucrarea poligoanelor este teorema lui *Jordan*, care spune că orice curbă plană închisă simplă împarte planul în două părţi: o parte interioară curbei, care este o regiune limitată, şi o parte exterioară curbei, care este o regiune nelimitată. Deşi pare simplă din punct de vedere intuitiv, demonstraţia teoremei lui Jordan este destul de dificilă şi poate fi găsită în referinţele bibliografice [Rour93].

Această teoremă justifică definiţia care se mai foloseşte pentru poligoane, şi anume: poligonul este o regiune limitată din plan, mărginită de o colecţie de segmente orientate consistent. Prin această definiţie se consideră poligonul ca o regiune închisă din plan. Uneori, poligonul este considerat ca fiind format numai din conturul său, deci numai de segmentele de dreaptă care mărginesc regiunea, şi nu de regiunea însăşi. În continuare, se foloseşte notaţia ∂P pentru a desemna conturul poligonului P (linia poligonală închisă care mărgineşte poligonul).

Triangularizarea poligoanelor. O altă proprietate importantă a poligoanelor este proprietatea de triangularizare. Se demonstrează că orice poligon poate fi împărţit în triunghiuri prin adăugarea a zero sau mai multe diagonale. Proprietatea de triangularizare se bazează pe noţiunile de vizibilitate şi diagonală în poligoane. Un punct x din interiorul unui poligon este vizibil pentru un alt punct y, dacă şi numai dacă segmentul xy nu este în nici un punct al său exterior poligonului, adică $xy \subseteq P$. Acest lucru înseamnă că linia care uneşte două puncte x şi y, vizibile unul altuia, poate atinge un vârf al poligonului. Vizibilitatea între două puncte x şi y este completă (*clearly visible*), dacă linia care uneşte cele două puncte nu atinge frontiera poligonală (fig. 2.2).

O diagonală a unui poligon este un segment de dreaptă între două vârfuri a şi b, complet vizibile unul altuia. Acest lucru înseamnă că intersecţia dintre segmentul închis ab şi ∂P este mulţimea {a,b}, adică segmentul ab nu atinge linia poligonală ∂P în alte puncte decât vârfurile a şi b, de început şi de sfârşit ale segmentului. Condiţiile ca segmentul ab care uneşte vârfurile a şi b ale unui poligon să fie o diagonală în acel poligon sunt deci: $ab \subseteq P$ şi $ab \cap \partial P = \{a, b\}$. Orice diagonală împarte un poligon în două poligoane mai mici (fig. 2.2).

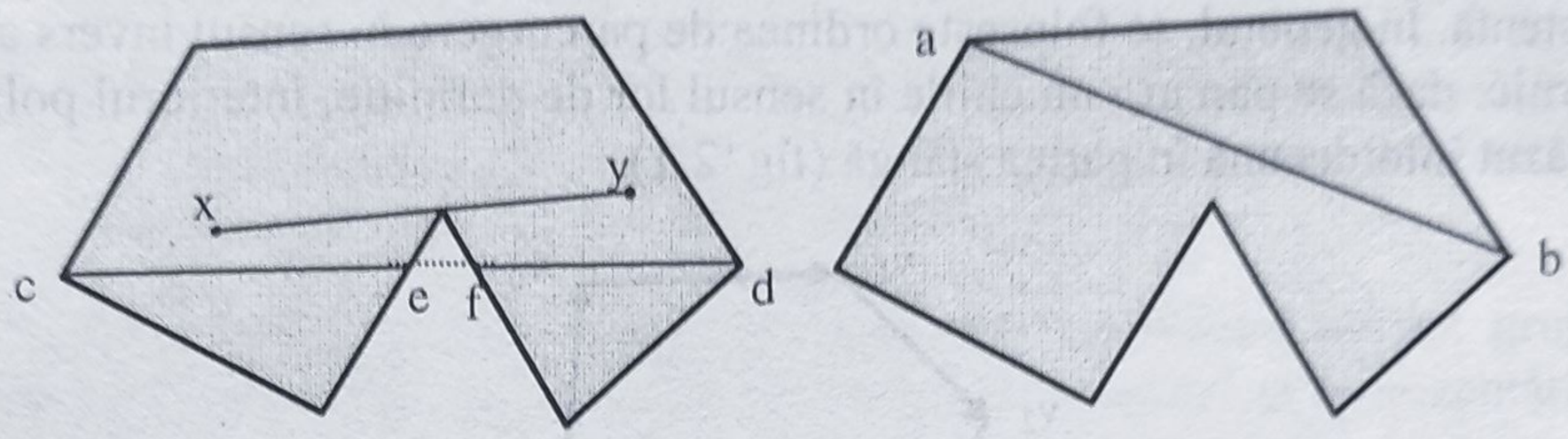


Fig. 2.2 Vizibilitate şi diagonale în poligoane.
Punctele x şi y sunt vizibile unul altuia.
Punctele c şi d sunt invizibile unul altuia deoarece ef ⊄ P.
Punctele a şi b sunt complet vizibile unul altuia, deci ab este diagonală în P.

Două diagonale ale unui poligon sunt neîncrucişate (*noncrossing*) dacă intersecţia lor este o submulţime a capetelor lor (punctele de început şi de sfârşit ale segmentelor). Dacă se adaugă atâtea diagonale neîncrucişate câte sunt posibile într-un poligon, atunci poligonul este împărţit în triunghiuri. O astfel de partiţionare a unui poligon în triunghiuri se numeşte *triangularizarea* poligonului. Diagonalele se pot adăuga în orice ordine, atât timp cât sunt neîncrucişate. Demonstraţia teoremei conform căreia orice poligon admite o triangularizare se bazează pe teorema lui Meister, demonstrată în [Rour93], care stabileşte că orice poligon cu $n \geq 4$ vârfuri admite cel puţin o diagonală.

Teorema triangularizării se bazează şi pe lema numărului de diagonale: Orice triangularizare a unui poligon P cu n vârfuri utilizează n – 3 diagonale şi constă din n – 2 triunghiuri. Aceste teoreme se demonstrează prin inducţie. În fig. 2.3 este prezentată triangularizarea unui poligon convex cu opt laturi; se inserează 8 – 3 = 5 diagonale neîncrucişate şi rezultă 8 – 2 = 6 triunghiuri.

Teorema triangularizării, care asigură că orice poligon poate fi divizat în triunghiuri, reprezintă suportul celei mai eficiente metode de generare (redare) a imaginii obiectelor tridimensionale: obiectele se reprezintă prin feţe poligonale, fiecare poligon se descompune în triunghiuri şi triunghiurile sunt generate prin algoritmi implementaţi hardware.

Din punct de vedere al reprezentării în program a poligoanelor, cea mai compactă formă este reprezentarea printr-o listă liniară de vârfuri, fiecare vârf fiind specificat printr-o structură (sau clasă, în programarea orientată pe obiecte) care memorează (cel puţin) coordonatele vârfului. Alte date referitoare la vârfurile poligoanelor necesare în modelarea şi redarea obiectelor (normală, culoare, coordonate de texturare, etc.) vor fi descrise în capitolele care urmează.

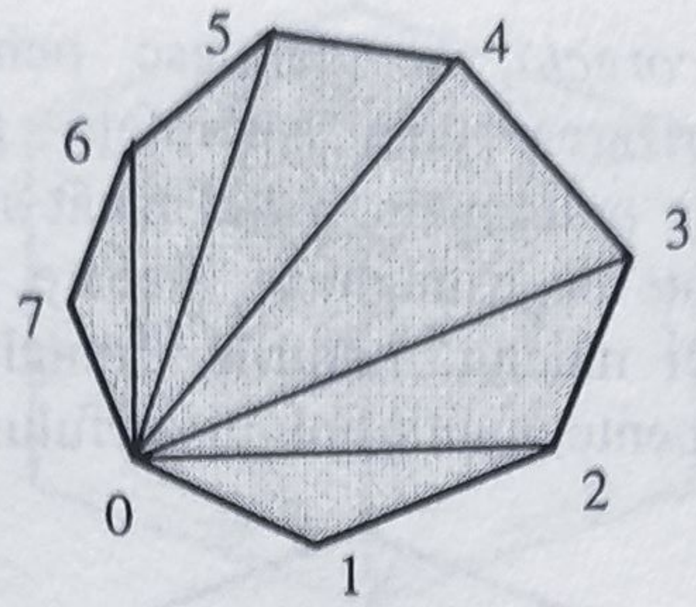


Fig. 2.3 Triangularizarea unui poligon convex.

Este posibilă reprezentarea unui poligon şi prin lista segmentelor sale, dar acestă reprezentare necesită un volum mai mare de date şi este folosită în implementarea anumitor algoritmi de prelucrare a poligoanelor (reuniune, divizare, etc.) şi mai puțin în reprezentarea modelului unui obiect. Lista liniară de vârfuri poate fi implementată ca vector sau ca listă simplu sau dublu înlănțuită.

2.1.2 REPREZENTAREA POLIEDRELOR

În modelarea şi reprezentarea prin suprafața de frontieră, obiectele sunt aproximate prin poliedre şi modelul lor este reprezentat prin suprafața poliedrului, compusă dintr-o colecție de poligoane.

Un poliedru reprezintă generalizarea în spațiul tridimensional a unui poligon din planul bidimensional: poliedrul este o regiune finită a spațiului a cărui suprafață de frontieră este compusă dintr-un număr finit de fețe poligonale plane. Suprafața de frontieră a unui poliedru conține trei tipuri de elemente geometrice: vârfurile (punctele), care sunt zero-dimensionale, muchiile (segmentele), care sunt unidimensionale şi fețele (poligoanele), care sunt bidimensionale (fig. 2.4).

Suprafața de frontieră a unui poliedru este o colecție finită de fețe poligonale care se intersectează corect. Intersecția corectă a fețelor înseamnă că, pentru fiecare pereche de fețe ale obiectului, fețele sunt disjuncte, au în comun un singur vârf, sau au în comun două vârfuri şi muchia care le uneşte.

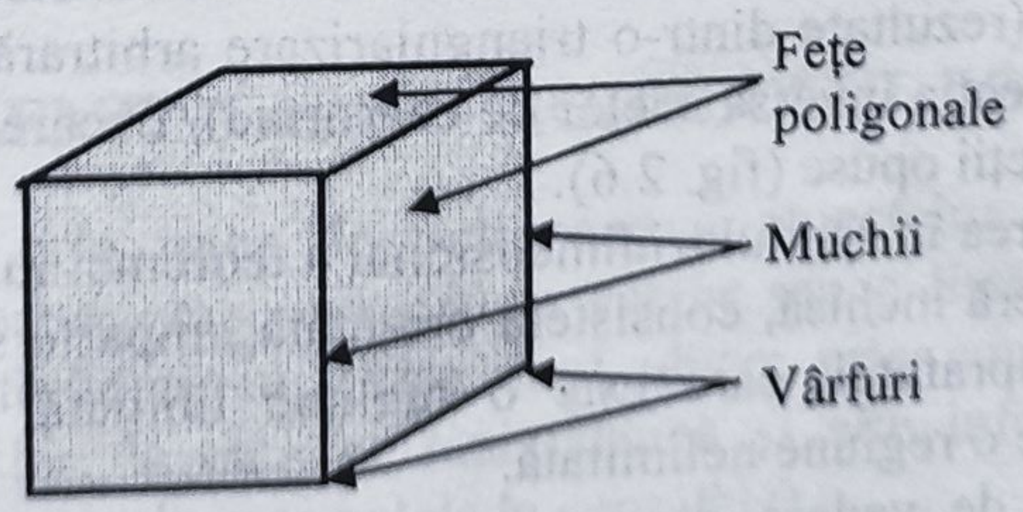


Fig. 2.4 Reprezentarea prin suprafața de frontieră a unui poliedru.

Din punct de vedere matematic, nu este imediat evident că un solid poate fi reprezentat univoc prin suprafața care îl mărgineşte. De aceea, este necesar să fie stabilite condițiile în care această reprezentare este permisă. Aceste condiții, numite

condiții de *construcție corectă*, se definesc pentru suprafețe de frontieră triangularizate. Triangularizarea unei suprafețe poliedrale se obține prin triangularizarea fiecărei fețe poligonale, astfel încât suprafața rezultată constă din vârfuri care sunt înconjurate de triunghiuri, fiecare pereche de triunghiuri fiind adiacente de-a lungul unei muchii. Laturile triunghiurilor adiacente unui vârf formează un circuit de segmente, numit link-ul vârfului (fig. 2.5).

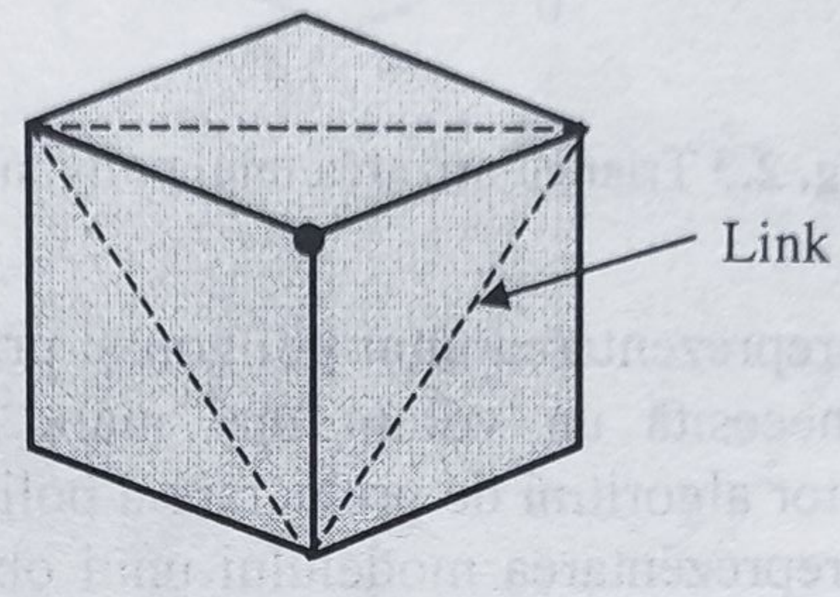


Fig. 2.5 Triangularizarea suprafeței de frontieră.
Link-ul unui vârf al suprafeței.

O suprafață de frontieră construită corect îndeplineşte următoarele condiții:

- Linkul fiecărui vârf al suprafeței triangularizate este complet, adică formează un circuit închis, nu neapărat planar.
- Triunghiurile suprafeței triangularizate sunt orientate consistent.

Închiderea link-ului fiecărui vârf asigură proprietățile suprafației de a fi *închisă* şi *conectată.* Proprietate de închidere înseamnă că suprafața nu are un sfârşit. Proprietatea de conectivitate înseamnă că există cel puțin o cale de la un triunghi la altul aparținând aceleiași suprafețe de frontieră, care poate fi parcursă prin traversarea muchiilor adiacente. Dacă o suprafață nu este închisă sau nu este conectată, prin triangularizarea suprafeței nu se obțin link-uri închise [Kal89].

Proprietate de construcție corectă a suprafeței de frontieră a unui solid se poate verifica folosind condiția de *orientare consistentă* a suprafeței, formulată de legea lui Moebius: O suprafață închisă este orientată consistent dacă la traversarea triunghiurilor sale (rezultate dintr-o triangularizare arbitrară), într-o direcție unică (de exemplu, în direcția inversă acelor de ceasornic), fiecare muchie este traversată de două ori, în direcții opuse (fig. 2.6).

Generalizarea în spațiul tridimensional a teoremei lui Jordan spune că orice suprafață de frontieră închisă, consistent orientată, împarte spațiul în două părți: o parte interioară suprafeței, care este o regiune limitată, şi o parte exterioară suprafeței, care este o regiune nelimitată.

Din punct de vedere geometric, orientarea consistentă se verifică prin direcția normalelor la fețele obiectului: dacă normalele fețelor sunt îndreptate către aceeaşi regiune a spațiului (fie toate îndreptate spre interior, fie toate îndreptate spre exterior), atunci suprafața are o orientare consistentă. Acest mod de verificare se referă la obiectele tridimensionale fără cavități, dar se poate extinde cu uşurință şi la obiecte care prezintă cavități.

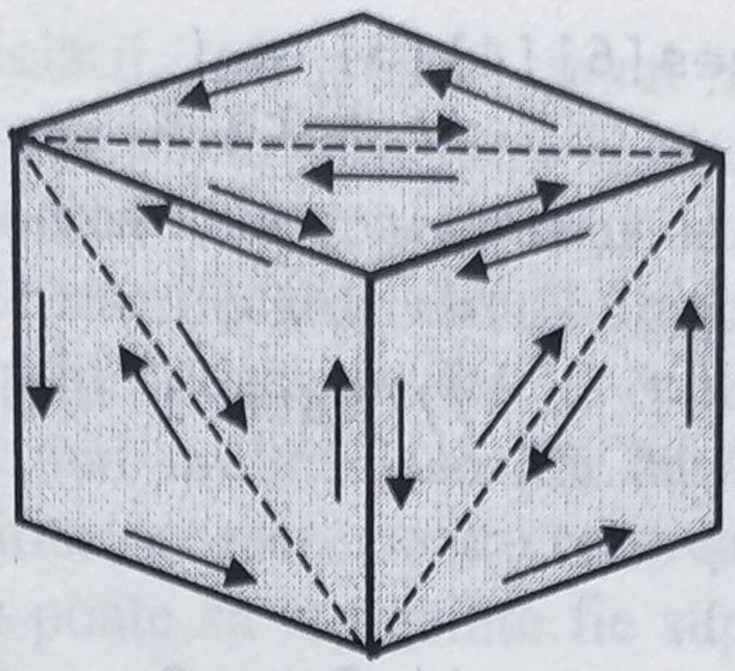


Fig. 2.6 Legea lui Mobius verifică orientarea consistentă suprafețelor triangularizate.

Teoretic, orientarea consistentă se verifică pentru fețele triangularizate ale suprafeței de frontieră, dar, prin extindere, se poate verifica orientarea consistentă folosind normalele la fețele poligonale, deoarece toate triunghiurile obținute prin triangularizarea unui poligon care reprezintă o față a unui obiect au aceeaşi orientare (fig. 2.7).

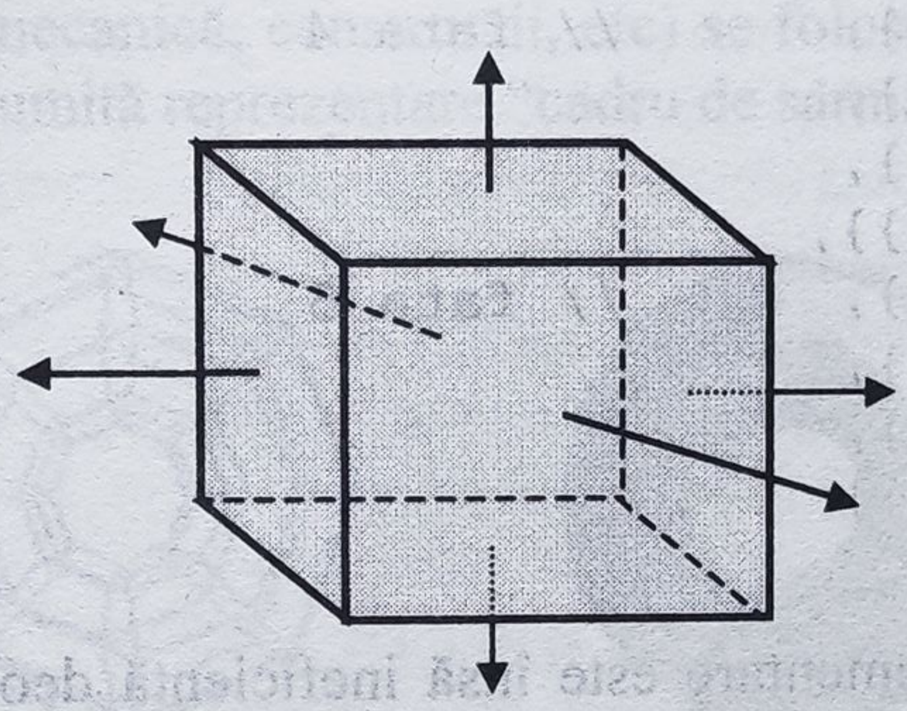


Fig. 2.7 Orientarea consistentă a fețelor obiectului.

Orientarea consistentă a fețelor poligonale ale obiectelor este o condiție de verificare a construcției corecte a suprafeței de frontieră şi, în acelați timp, este folosită în operațiile de eliminare a suprafețelor ascunse în cursul redării obiectelor tridimensionale. De aceea, normalele la fețele poligonale se memorează în modelul obiectelor, împreună cu coordonatele vârfurilor.

2.1.3 IMPLEMENTAREA MODELULUI POLIGONAL

Reprezentarea prin rețea de poligoane a obiectelor se implementează printr-o listă de poligoane, care poate fi un vector sau o listă înlănțuită. Astfel, întreaga informație referitoare la forma unui obiect este compusă din liste de coordonate ale vârfurilor, la care se mai adaugă şi alte informații geometrice necesare în redare (de exemplu, normalele la suprafețe).

Fără să se insiste acum asupra eficienței sau eleganței implementării (se va reveni ulterior), se poate considera că forma unui obiect modelat prin rețea de poligoane poate fi reprezentată ca un vector de fețe, fiecare față fiind un vector de vârfuri, fiecare vârf fiind un vector de trei coordonate în virgulă flotantă. De exemplu, un cub poate fi reprezentat astfel:

```
double CubeFaces[6][4][3] = {
    {{-1,-1,-1},        // fata 0
     { 1,-1,-1},
     { 1,-1, 1},
     {-1,-1, 1}},
    {{-1, 1, 1},        // fata 1
     { 1, 1, 1},
     { 1, 1,-1},
     {-1, 1,-1}},
    {{-1,-1, 1},        // fata 2
     { 1,-1, 1},
     { 1, 1, 1},
     {-1, 1, 1}},
    {{ 1,-1,-1},        // fata 3
     {-1,-1,-1},
     {-1, 1,-1},
     { 1, 1,-1}},
    {{ 1,-1, 1},        // fata 4
     { 1,-1,-1},
     { 1, 1,-1},
     { 1, 1, 1}},
    {{-1,-1, 1},        // fata 5
     {-1, 1, 1},
     {-1, 1,-1},
     {-1,-1,-1}}
};
```

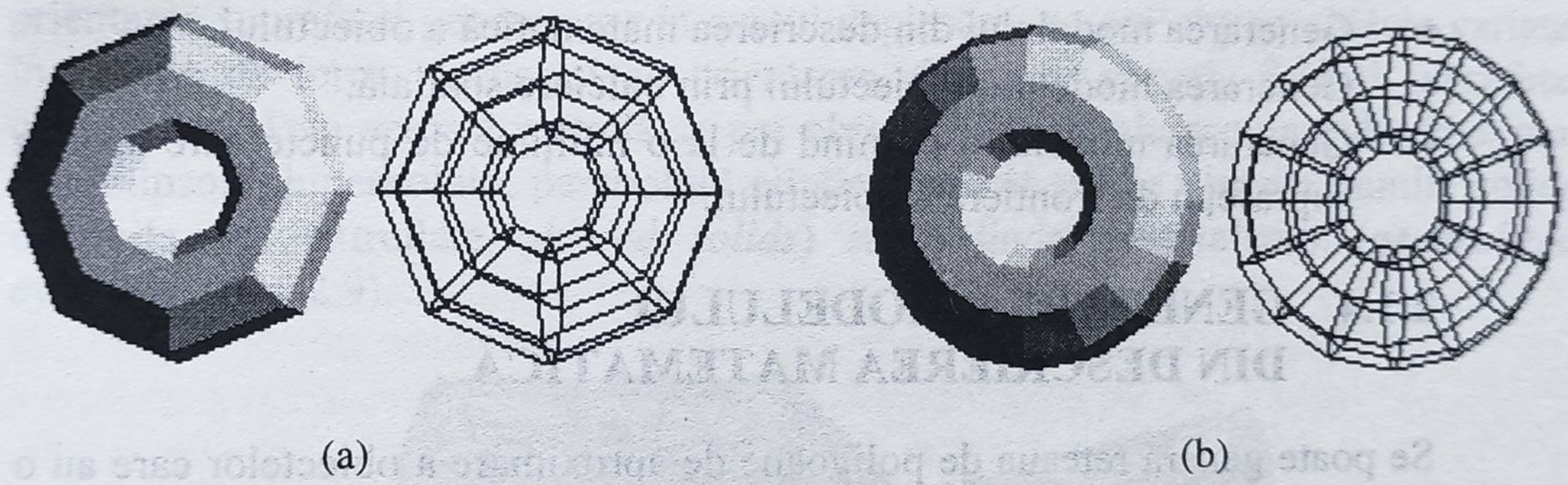
O astfel de implementare este însă ineficientă deoarece fiecare vârf este prelucrat de trei ori, pentru fiecare față adiacentă acestuia. O implementare mult mai eficientă a modelului poligonal definește un vector cu toate vârfurile unui obiect, iar fiecare față se definește printr-un vector de indecși în vectorul de vârfuri. De exemplu, modelul unui cub folosind indecși pentru definirea fețelor arată astfel:

```
double CubeCoords[8][3]={
        {-1,-1, 1},
        { 1,-1, 1},
        { 1,-1,-1},
        {-1,-1,-1},
        {-1, 1, 1},
        { 1, 1, 1},
        { 1, 1,-1},
        {-1, 1,-1}
};
int CubeIndexFace[6][4]={
        {3, 2, 1, 0},
        {4, 5, 6, 7},
        {0, 1, 5, 4},
        {2, 3, 7, 6},
        {1, 2, 6, 5},
        {0, 4, 7, 3}
};
```

Atât în cursul modelării, cât şi în cursul redării imaginii, modelul obiectelor se reprezintă mai complex, conținând și alte informații geometrice și atribute de aspect. Astfel, culoarea (mai general spus, materialul) este o informație care se asignează fiecărei fețe sau fiecărui vârf al obiectului. În general, modelul poligonal cu reprezentarea prin indecşi a feţelor poligonale este implementat orientat pe obiecte, folosind mai multe clase (de bază şi derivate) care permit încapsularea tuturor informațiilor necesare pentru redarea obiectelor.

O rețea de poligoane poate să reprezinte fie suprafața de frontieră a unui solid, fie o suprafață deschisă, necesară în anumite situații de modelare, cum este suprafața terenului simulat într-o scenă virtuală. Deoarece se modelează o zonă geografică limitată şi nu se explorează scena astfel ca să fie văzută marginea "pământului", se poate folosi o suprafață poligonală deschisă pentru reprezentarea terenului.

Reprezentarea obişnuită în grafica din scenele virtuale este aceea în care fiecare poligon este reprezentat ca o suprafață (poligoane "pline"), dar în proiectările grafice (în mecanică, construcții, etc) se foloseşte şi reprezentarea prin contur a poligoanelor, numită reprezentare "cadru de sârmă" (*wireframe*) (fig. 2.8).



(a) (b)

Fig. 2.8 Reprezentarea "plină" (*filled*) şi "cadru de sârmă" (*wireframe*) a obiectelor.

Precizia de reprezentare a modelului, adică diferența dintre suprafața obiectului şi fețele poligonale prin care este aproximat, este un parametru important de modelare. În general, cu cât numărul de poligoane prin care se aproximează suprafața obiectului este mai mare, cu atât precizia de reprezentare este mai bună. Se observă diferența dintre imaginea unui obiect (tor) reprezentat prin 64 de poligoane în fig. 2.8(a) şi prin 128 de poligoane în fig. 2.8 (b).

Folosirea unui număr mare de poligoane pentru reprezentarea obiectelor conduce la un volum imens de date în modelarea scenelor virtuale, care implică cerințe corespunzătoare de memorare şi de capacitate de calcul. Oricât de mult ar creşte capacitatea de prelucrare în sistemele grafice, prin folosirea paralelismului şi prin implementarea hardware a algoritmilor de prelucrare, performanțe de generare interactivă în timp real a imaginilor se pot obține numai dacă obiectele scenei se modelează în mod corespunzător. Metodele de aproximare eficientă a modelelor sunt cunoscute sub numele de *simplificarea* (sau *rafinarea*) *datelor*.

Tehnica de simplificare directă, prin reducerea uniformă a numărului de poligoane de reprezentare a obiectului, este inacceptbilă, deoarece nu se poate controla precizia de reprezentare. Pentru păstrarea unei precizii de reprezentare constante, se foloseşte modelarea adaptivă, în care dimensiunea feţelor poligonale variază în funcţie de curbura suprafeţei; în părţile cu curbură pronunţată sunt introduse mai multe poligoane pe unitatea de suprafaţă.

O altă metodă de simplificare a datelor este modelarea cu nivele de detaliu multiple a obiectelor (*levels of detail*- LOD), în care modelul unui obiect este compus dintr-o succesiune de reprezentări, fiecare cu o precizie diferită. În cursul generării imginilor, se selectează nivelul de detaliu corespunzător, în funcţie de poziţia obiectului faţă de punctul de observare. Cu cât obiectul este mai depărtat, se selectează o reprezentare cu o precizie mai scăzută a obiectului. Acestă metodă este posibilă datorită folosirii proiecţiei perspectivă în generarea imaginilor şi va fi înţeleasă mai uşor după parcurgerea capitolului următor privind transformările geometrice.

Modelul poligonal al unui obiect se poate genera prin mai multe metode, în funcţie de tipul obiectului şi de aplicaţia grafică în care este folosit modelul respectiv. Se poate folosi una din următoarele metode de modelare poligonală:

- Generarea modelului din descrierea matematică a obiectului.
- Generarea modelului obiectului prin baleiere spaţială.
- Generarea modelului pornind de la o mulţime de puncte care aparţin suprafeţei de frontieră a obiectului.

2.1.4 GENERAREA MODELULUI DIN DESCRIEREA MATEMATICĂ

Se poate genera reţeaua de poligoane de aproximare a obiectelor care au o descriere matematică cunoscută. De exemplu, ecuaţiile unor suprafeţe quadrice:

- Elipsoid:

$$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0 \qquad (2.1)$$

unde a, b c sunt semiaxele elipselor.

- Hiperboloid:

$$x^2/a^2 + y^2/b^2 - z^2/c^2 - 1 = 0 \quad \text{si} \quad x^2/a^2 + y^2/b^2 - z^2/c^2 + 1 = 0 \qquad (2.2)$$

- Paraboloid eliptic:

$$x^2/a^2 + y^2/b^2 = z \qquad (2.3)$$

Surprafaţa se intersectează mai întâi cu un număr n de plane perpendiculare pe axa Oz, de ecuaţii $z = n\,\Delta z$, pentru $n = -k, -k+1, \ldots, -1, 0, 1, 2, \ldots, k-1, k$. Se obţin n elipse (paralele) şi pe fiecare elipsă se eşantionează m

puncte echidistante (pe meridiane), obținându-se (n–1)×m poligoane care aproximează suprafața de frontieră a obiectului respectiv.

Aceste suprafețe se pot obține şi prin rotația unei curbe în jurul unei axe de rotație. De exemplu, suprafața elipsoidului se obține prin rotația în jurul axei z a elipsei:

$$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0, \quad y = 0. \tag{2.4}$$

Prin rotația unei curbe în jurul unei axe se pot obține obiecte tridimensionale mai variate, în funcție de forma curbei care se roteşte. De exemplu, un tor se obține prin rotația unui cerc în jurul unei axe paralele cu planul cercului. Suprafețele astfel obținute se numesc suprafețe de rotație.

2.1.5 GENERAREA MODELULUI PRIN BALEIERE SPAȚIALĂ

Se pot genera obiecte tridimensionale prin deplasarea (*sweeping*) unei suprafețe generatoare de-a lungul unei curbe oarecare. Dacă se variază forma şi orientarea suprafeței generatoare în cursul deplasării, se pot obține obiecte variate, în funcție de forma curbei şi de orientarea, forma şi variația formei suprafeței generatoare. Prin această metodă se pot obține atât formele regulate descrise mai sus (elipsoid, hiperboloid, paraboloid eliptic, tor) cât şi alte obiecte numite solide ductibile sau extrudate (*ducted solids*) sau cilindri generalizați (*generalized cylinders*) (fig. 2.9).

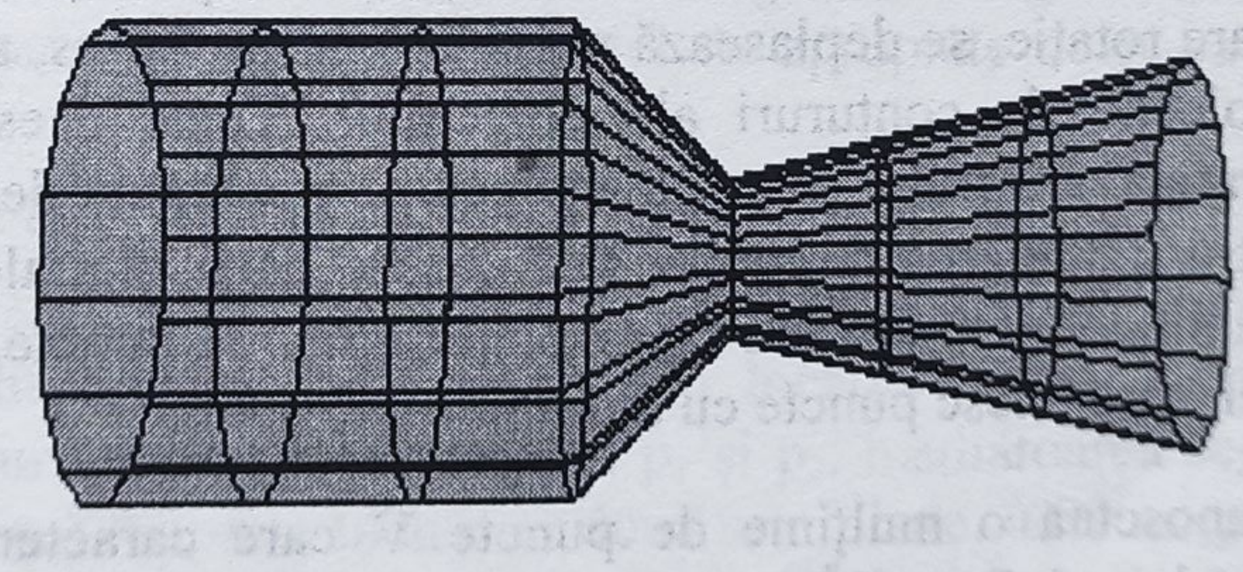


Fig. 2.9 Obiect poligonal modelat prin deplasarea unui cerc cu diametru variabil de-a lungul unei linii drepte.

Pentru definirea deplasării unei suprafețe de-a lungul unei curbe, este necesar să se definească intervalul curbei pe care are loc deplasarea şi modul în care se divide intervalul parcurs. Împărțirea intervalului în distanțe egale nu dă rezultate bune, deoarece punctele obținute nu vor fi egal distribuite pe suprafața obiectului. De aceea este necesară divizarea intervalului în funcție de curbura curbei. Dacă curbura este pronunțată, se aleg subdiviziuni mai mici ale intervalului, iar pentru curburi mai reduse se aleg subdiviziuni mai mari ale intervalului.

2.1.6 GENERAREA MODELULUI PE BAZA UNEI MULȚIMI DE PUNCTE CARE APARȚIN SUPRAFEȚEI DE FRONTIERĂ A OBIECTULUI

O metodă mai generală de modelare a obiectelor tridimensionale se bazează pe cunoaşterea unei mulțimi de puncte distribuite uniform sau neuniform pe suprafața obiectelor. Această metodă implică mai întâi obținerea colecției de puncte şi apoi construirea rețelei de poligoane care să aproximeze obiectul cu precizia dorită. Dacă metodele prezentate mai înainte pot fi utilizate pentru o categorie restrânsă de obiecte, în schimb, metoda generării modelului pornind de la o mulțime de puncte ale suprafeței acestuia poate fi aplicată pentru cele mai variate obiecte: clădiri, vehicule, plante, animale, elemente anatomice, teren, etc.

Mulțimea de puncte de pe suprafața obiectului se determină diferit, în funcție de modul în care este cunoscut sau reprezentat obiectul real:

- Pentru obiectele care se proiectează într-un sistem de proiectare asistat de calculator – CAD (*Computer Aided Design*), informațiile de formă necesare în generearea modelului se pot obține din reprezentarea proiectului. De exemplu, clădiri, obiecte mecanice, vehicule pot fi proiectate în AutoCad şi coordonatele vârfurilor suprafețelor sunt folosite pentru reprezentarea prin rețea poligonală a obiectului.
- Pentru obiectele reale, sau machete ale acestora, se poate folosi un scanner 3D cu laser. Obiectul real (sau macheta acestuia) este plasat pe o masă rotativă în direcția de emisie a unei raze laser. La fiecare rotație completă a mesei se obține o colecție de puncte pe un contur al obiectului, prin măsurarea distanței la suprafața obiectului. După fiecare rotație, se deplasează masa în sus sau în jos, astfel că se obține o colecție de contururi ale obiectului. Toate aceste puncte de pe suprafața obiectului sunt folosite pentru crearea modelului poligonal.
- Pentru modelarea terenului se folosesc hărți digitale care furnizează altitudinile terenului într-o grilă uniformă de puncte, sau contururi de nivel, care unesc puncte cu altitudine constantă.

Fiind cunoscută o mulțime de puncte V care caracterizează un obiect tridimensional, volumul $\Omega \subset R^3$ acoperit de aceste puncte se numeşte domeniul mulțimii V. Ω este un poliedru, convex sau neconvex, iar punctele mulțimii V pot fi dispuse regulat sau neregulat în domeniul Ω. Cazul cel mai frecvent întâlnit în modelare este acela în care punctele mulțimii V sunt distribuite neregulat în spațiu şi aproximarea obiectului se realizează printr-o rețea tridimensională de poligoane Γ, care aproximează liniar pe porțiuni obiectul dat. Rețeaua Γ trebuie să fie construită pornind de la mulțimea V de puncte în spațiu şi de la cerințele de precizie de aproximare a obiectului. Construirea modelului poligonal al unui obiect cunoscut printr-o mulțime de puncte de pe suprafața sa se poate realiza prin mai multe metode, dintre care cele mai folosite sunt triangularizarea (*triangulation*) şi transformata wavelet [Gross96]. În continuare sunt prezentate proprietățile şi principiile de realizare a triangularizării.

2.1.7 TRIANGULARIZAREA UNEI MULȚIMI DE PUNCTE

Triangularizarea în două sau trei dimensiuni a fost intens studiată, începând cu Dirichlet, Voronoi şi continuând cu Delaunay. Un număr mare de cărți şi articole tratează proprietățile triangularizării şi algoritmii de construcție a modelelor prin triangularizare [Aur91], [Rour93]. Dintre diferitele modalități de triangularizare, triangularizarea Delaunay are proprietatea că unghiurile minime ale triunghiurilor generate sunt maxime față de toate celelate triangularizări. Această proprietate este deosebit de utilă în grafică, unde suprafețele ascuțite (cu unghiuri mici) provoacă un zgomot (zgomot de aliasing), care creşte cu cât scade dimensiunea primitivelor geometrice (triunghiurile).

Triangularizarea Delauny este o structură duală uneia din cele mai importante structuri din geometrie, numită diagrama Voronoi. Conceptul de diagramă Voronoi are peste un secol vechime, fiind descris de Dirichlet în 1850 şi de Voronoi în 1908. Diagrama Voronoi are numeroase aplicații în diferite domenii ale ştiinței şi tehnicii (codare, robotică, cristalografie, etc).

2.1.7.1 Diaragama Voronoi în plan

Fie o mulțime de puncte $P = \{p_1, p_2, \ldots, p_n\}$ în spațiul euclidian bidimensional. Aceste puncte se numesc locații (*sites*). Se partiționează planul bidimensional prin asignarea fiecărui punct din plan celei mai apropiate locații, p_i. Toate aceste puncte formează *regiunea Voronoi a locației* $R(p_i)$. (Regiunea Voronoi se mai numeşte şi domeniul Dirichlet, regiunea Wigner-Seitz sau polinomul Tiessen; această regiune nu este, însă, poligon în sensul definiției date în paragraful precedent, deoarece poate fi nemărginită). Regiunea $R(p_i)$ constă din mulțimea tuturor punctelor cel puțin la fel de apropiate de p_i ca de oricare altă locație:

$$R(p_i) = \{x: |p_i - x| \leq |p_i - x|, \forall\, j \neq i\} \qquad (2.5)$$

Unele puncte din plan nu au o singură locație cea mai apropiată (vecină). Mulțimea punctelor care au mai mult de o locație vecină formează diagrama Voronoi V(P). Pentru două puncte în plan p_1 şi p_2, mediatoarea segmentului p_1p_2, notată $M_{12}= M(p_1, p_2)$, împarte planul în două semiplane, unul asignat punctului p_1, celălalt asignat punctului p_2 (fig. 2.10(a)). Pentru mai mult de două puncte în plan, se consideră mediatoarea M_{ij} a fiecărei perechi de puncte p_i şi p_j, şi $H(p_i, p_j)$ semiplanul închis de dreapta M_{ij} care conține locația p_i.

Dat fiind că regiunea Voronoi a unei locații $R(p_i)$ este mulțimea tuturor punctelor mai apropiate de p_i decât de oricare altă locație, înseamnă că ea conține toate punctele mai apropiate de p_i decât de p_1 *şi* mai apropiate de p_i decât de p_2 *şi* mai apropiate de p_i decât de p_3 şi aşa mai departe. Se poate scrie următoarea ecuație pentru $R(p_i)$:

$$R(p_i) = \bigcap_{i \neq j} H(p_i, p_j) \qquad (2.6)$$

Intersecția semiplanelor $H(p_i, p_j)$ se calculează pentru orice $i \neq j$ şi rezultă diagrama Voronoi a mulțimii de puncte date (fig. 2.10(b)

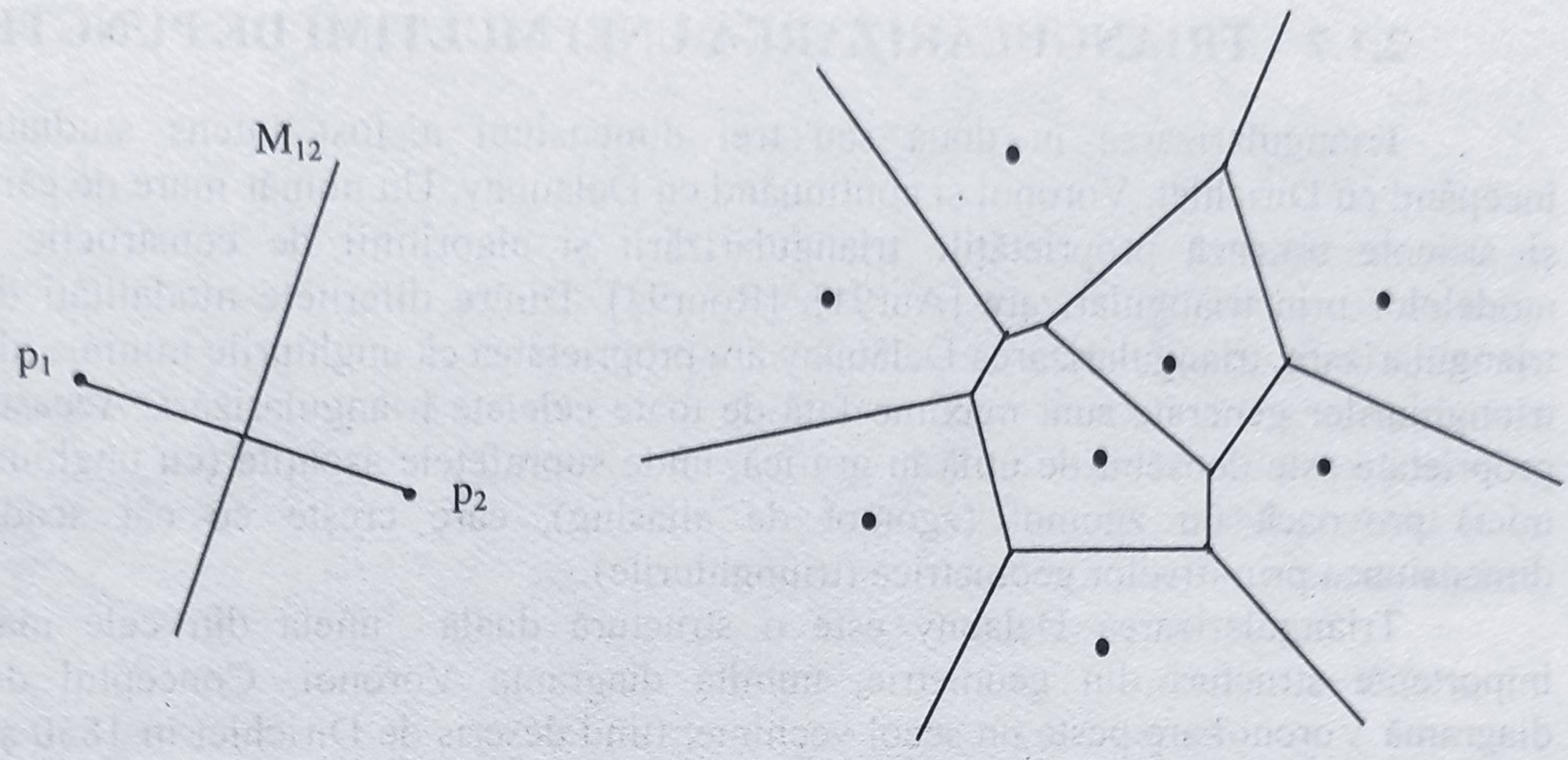


Fig. 2.10 Diagrame Voronoi.
(a) Pentru două locații (b) Pentru n = 6 locații.

Diagrama Voronoi a n puncte conține n regiuni Voronoi şi toate sunt regiuni convexe, dat fiind că intersecția dintre oricâte semiplane este o mulțime convexă de puncte în plan. Dacă regiunile sunt limitate, ele sunt poligoane convexe. În diagrama Voronoi, muchiile se numesc muchii Voronoi, iar intersecțiile între muchii definesc vârfurile Voronoi. Oricare punct de pe muchiile Voronoi are două locații vecine, iar oricare vârf are cel puțin trei locații vecine.

2.1.7.2 Triangularizarea Delaunay în plan

Se defineşte graful dual al unei diagrame Voronoi V(P) drept graful G care se construieşte în felul următor: nodurile grafului sunt locațiile diagramei V(P) şi două noduri în graf sunt conectate între ele dacă regiunile Voronoi corespunzătoare locațiilor acestora au o muchie Voronoi comună. În 1934 Delaunay a demonstrat că, dacă muchiile grafului dual al diagramei Voronoi se reprezintă prin segmente de dreaptă şi nu există patru locații conciclice, atunci se obține o triangularizare plană a locațiilor P. Această triangularizare se numeşte triangularizare Delaunay, notată D(P) (fig. 2.11). Cazul cu patru locații conciclice este studiat şi tratat separat.

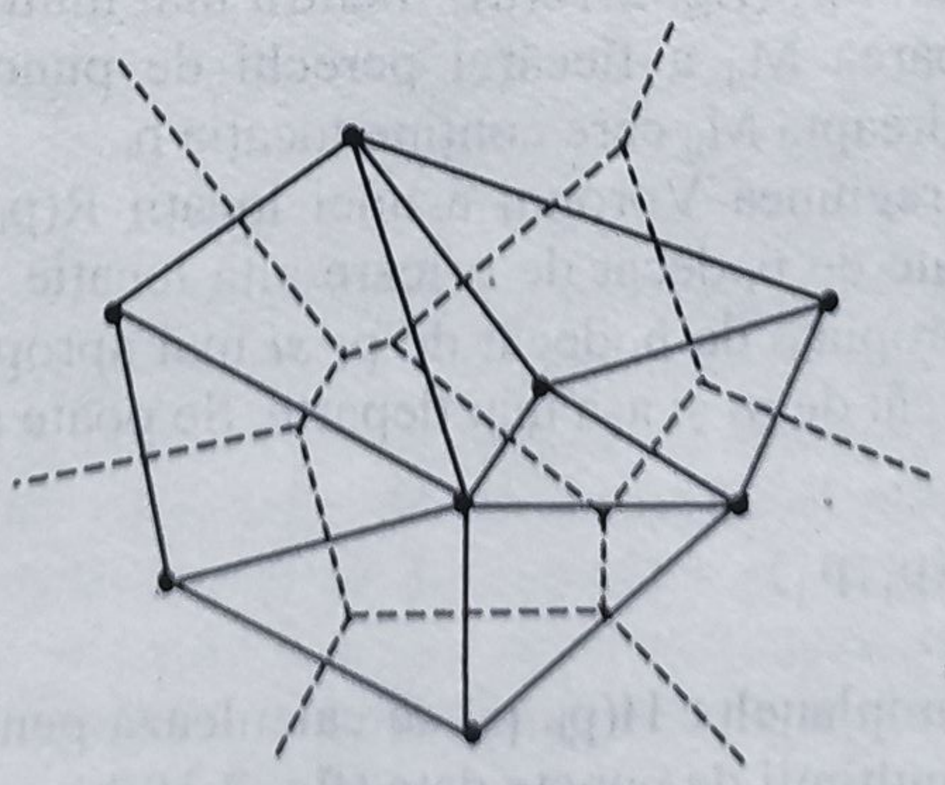


Fig. 2.11 Triangularizarea Delaunay este graful dual al diagramei Voronoi.

Diagrama Voronoi şi triangularizarea Delaunay sunt structuri duale, care conţin aceleaşi informaţii, dar reprezentate în forme diferite. Câteva din proprietăţile acestor structuri sunt:

- Fiecare triunghi din D(P) corespunde unui vârf în V(P).
- Fiecare muchie din D(P) corespunde unei muchii în V(P).
- Fiecare nod (vârf) din D(P) corespunde unei locaţii în V(P).
- Frontiera lui D(P) reprezintă acoperirea convexă a punctelor.
- În interiorul fiecărui triunghi din D(P) nu există nici o locaţie.
- Fiecare regiune Voronoi $V(p_i)$ este convexă.
- Regiunea $V(p_i)$ este nelimitată dacă şi numai dacă p_i se află pe frontiera (acoperirea convexă) triangularizării Delaunay.
- Dacă v este un vârf Voronoi la intersecţia regiunilor $V(p_1)$, $V(p_2)$, $V(p_3)$, atunci v este centrul cercului C(v), determinat de p_1, p_2, p_3. Această proprietate se generalizează pentru vârfuri de orice grad.
- Cercul C(v), circumscris triunghiului Delaunay corespunzător vârfului v, nu conţine nici o locaţie în interiorul lui.
- Dacă p_j este cel mai apropiat vecin al lui p_i, atunci (p_i, p_j) este o muchie în D(P).
- Dacă există un cerc care trece prin p_i şi p_j şi nu conţine nici o locaţie în interiorul său, atunci (p_i, p_j) este o muchie în D(P). Inversa acestei proprietăţi este: pentru fiecare muchie Delaunay există un cerc vid (care nu conţine nici o locaţie în interiorul său).

Demonstraţia acestor proprietăţi se poate găsi în mai multe lucrări [Rour93], [Prep85]. Ultima proprietate este cunoscută sub numele de *proprietatea cercului vid* (*empty circle property*) şi este una din metodele de triangularizare Delaunay. Numeroasele aplicaţii şi frumuseţea structurilor Voronoi şi Delaunay au condus la cercetări intense pentru elaborarea algoritmilor de generare a acestora. În grafică interesează cel mai mult triangularizarea unei mulţimi de puncte. Triangularizarea se poate construi direct sau prin intermediul diagramei Voronoi, care, uneori, se obţine mai uşor. Pornind de la diagrama Voronoi, triangularizarea Delaunay se obţine construind câte o muchie pentru fiecare pereche de vârfuri Voronoi vecine (care au o muchie Voronoi comună).

Metoda cercului vid. Metoda cea mai directă de construire a triangularizării Delaunay a unei mulţimi P de puncte este cea bazată pe proprietatea cercului vid: pentru fiecare triplet (p_i, p_j, p_k) se testează incluziunea oricărei alte locaţii: dacă cercul respectiv nu conţine nici o locaţie, atunci muchiile triunghiului $(p_i\ p_j\ p_k)$ aparţin triangularizării D(P). Acest algoritm este extrem de simplu de realizat (conţine trei bucle imbricate în care se testează incluziunea unui punct într-un cerc), dar este unul dintre cei mai lenţi algoritmi de triangularizare, având o complexitate în $O(n^3)$ şi nu poate fi folosit decât pentru număr redus de puncte.

Algoritmul de triangularizare divide-and-conquer. Algoritmul cel mai rapid de construire a diagramei Voronoi, din care se poate obţine imediat

triangularizarea Delaunay, este algoritmul de tip împarte-şi-cucereşte (*divi*
conquer) definit de Shamos şi Hoey în 1975, cu o complexitate în O(*n*
Acest algoritm este însă deosebit de dificil de implementat şi de aceea sunt
algoritmi mai puțin eficienți, dar mai simpli.

Algoritmul incremental. Unul dintre cei mai populari algori
triangularizare este algoritmul incremental, care are o complexitate în $O(n^2$
dată triangularizarea Delaunay D a unei mulțimi de *k* puncte, se poate c
triangularizarea D' pentru mulțimea de puncte inițială la care se adaugă î
punct p. Pentru inserarea unui nou punct p într-o triangularizare se determi
întâi regiunea R a noului punct, formată din mulțimea triunghiurilor al căr
circumscris conține punctul p (fig. 2.12).

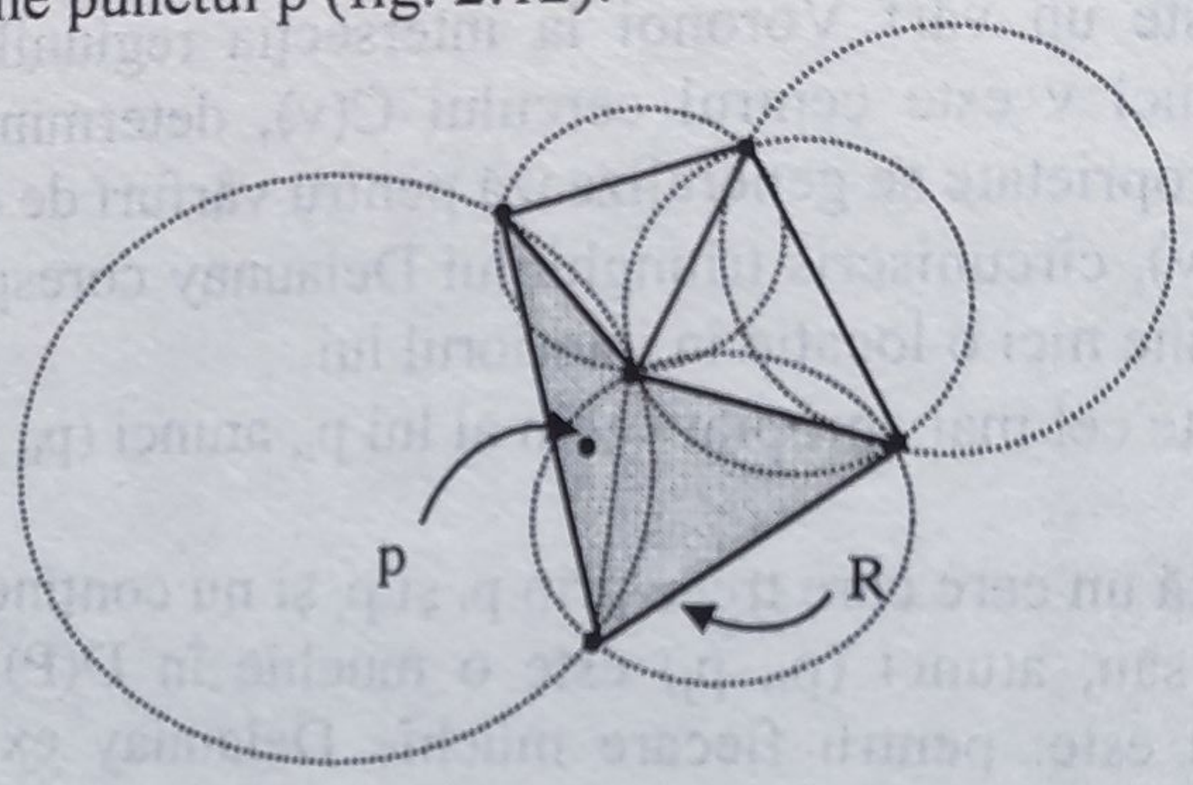


Fig. 2.12 Algoritmul incremental de triangularizare Delaunay.

Regiunea R are propietatea că nu conține nici un alt punct în interio
cu excepția lui p. Din această regiune se reține frontiera prin eliminarea t
muchiilor interne. După eliminarea muchiilor interne, toate vârfurile regiuni
vizibile din punctul p şi triangularizarea D' se obține prin unirea punctului
fiecare din vârfurile regiunii R.

2.1.7.3 Triangularizarea Delaunay în spațiul tridimensional

Se consideră o mulțime de puncte în spațiu (locații), (x_i, y_i, z_i), i = 1
Triangularizarea Delaunay în spațiu înseamnă umplerea domeniului ac
mulțimi de puncte cu tetraedre, astfel încât vârfurile tetraedrelor să fie punct
mulțimea dată şi sfera circumscrisă fiecărui tetraedru să nu conțină nici o
locație în interiorul ei. Această condiție este extinderea în spațiul tridimensio
condiției cercului vid din triangularizarea Delaunay în plan.

Algoritmii de construire a triangularizării Delaunay în sp
tridimensional sunt destul de complicați, necesitând testarea a numeroase sit
particulare care apar datorită capacității limitate de reprezentare a valo
coordonatelor [Fang95]. Rezultatul triangularizării Delaunay este o rețea (de
mai multe ori neregulată) de poligoane în spațiu, prin care se aproxim
suprafața de frontieră a obiectului respectiv. Aparatura modernă de scanare 3D
înglobați algoritmi de triangularizare tridimensională, astfel că utilizatorul prim
direct modelul poligonal al obiectului scanat.

triangularizarea Delaunay, este algoritmul de tip împarte-şi-cucereşte (*divide-and-conquer*) definit de Shamos şi Hoey în 1975, cu o complexitate în O(n log n). Acest algoritm este însă deosebit de dificil de implementat şi de aceea sunt folosiți algoritmi mai puțin eficienți, dar mai simpli.

Algoritmul incremental. Unul dintre cei mai populari algoritmi de triangularizare este algoritmul incremental, care are o complexitate în O(n^2). Fiind dată triangularizarea Delaunay D a unei mulțimi de k puncte, se poate construi triangularizarea D' pentru mulțimea de puncte inițială la care se adaugă încă un punct p. Pentru inserarea unui nou punct p într-o triangularizare se determină mai întâi regiunea R a noului punct, formată din mulțimea triunghiurilor al cărui cerc circumscris conține punctul p (fig. 2.12).

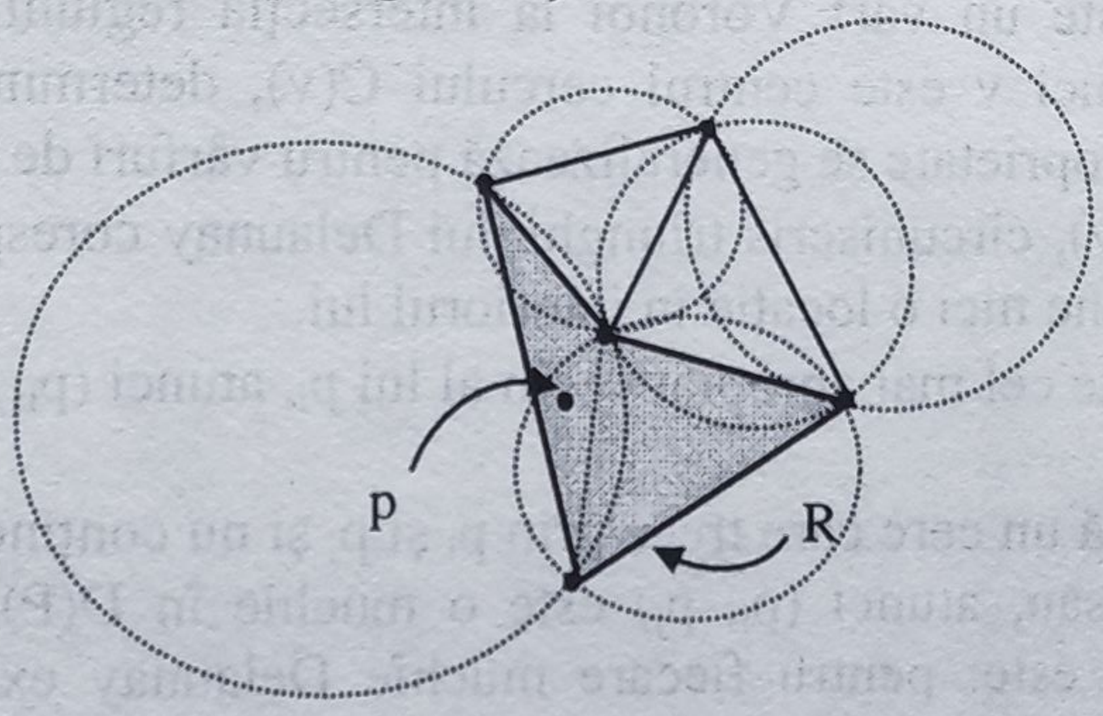


Fig. 2.12 Algoritmul incremental de triangularizare Delaunay.

Regiunea R are propietatea că nu conține nici un alt punct în interiorul ei, cu excepția lui p. Din această regiune se reține frontiera prin eliminarea tuturor muchiilor interne. După eliminarea muchiilor interne, toate vârfurile regiunii sunt vizibile din punctul p şi triangularizarea D' se obține prin unirea punctului p cu fiecare din vârfurile regiunii R.

2.1.7.3 Triangularizarea Delaunay în spațiul tridimensional

Se consideră o mulțime de puncte în spațiu (locații), (x_i, y_i, z_i), i = 1,...n. Triangularizarea Delaunay în spațiu înseamnă umplerea domeniului acestei mulțimi de puncte cu tetraedre, astfel încât vârfurile tetraedrelor să fie puncte din mulțimea dată şi sfera circumscrisă fiecărui tetraedru să nu conțină nici o altă locație în interiorul ei. Această condiție este extinderea în spațiul tridimensional a condiției cercului vid din triangularizarea Delaunay în plan.

Algoritmii de construire a triangularizării Delaunay în spațiul tridimensional sunt destul de complicați, necesitând testarea a numeroase situații particulare care apar datorită capacității limitate de reprezentare a valorilor coordonatelor [Fang95]. Rezultatul triangularizării Delaunay este o rețea (de cele mai multe ori neregulată) de poligoane în spațiu, prin care se aproximează suprafața de frontieră a obiectului respectiv. Aparatura modernă de scanare 3D are înglobați algoritmi de triangularizare tridimensională, astfel că utilizatorul primeşte direct modelul poligonal al obiectului scanat.

2.1.7.4 Triangularizarea suprafețelor de teren

Un caz special de triangularizare tridimensională este triangularizarea suprafețelor de teren. În aplicațiile care necesită reprezentarea terenului unei zone geografice se pot folosi coordonate geocentrice, coordonate geodezice sau coordonate într-un sistem de proiecție cartografic cunoscut (UTM – *Universal Transversal Mercator*, *Lambert*, etc).

În general, toate aceste sisteme de coordonate folosesc datele de referință ale globului pământesc standard WGS84. Sistemul de coordonate geocentric este un sistem de coordonate tridimensional cu originea în centrul Pământului. Coordonatele geodezice sunt coordonalele date în latitudine şi longitudine. Coordonatele UTM se obțin prin proiecția suprafeței Pământului pe un cilindru care înfăşoară globul pământesc de-a lungul unui anumit meridian. Cilindrul Mercator este desfăşurat într-un plan în care este definit un sistem de referință bidimensional (x, y), cu axa x către Sud şi axa y către Est, la care se adaugă axa z de coordonate pentru altitudine, pentru obținerea unui sistem de coordonate tridimensional drept. În coordonate UTM, eroarea de proiecție creşte cu cât creşte distanța față de meridianul de definiție al proiecției.

Sistemul de coordonate terestre (geocentric, geodezic sau UTM) este considerat sistem de referință universal în reprezentarea scenelor virtuale. În unele sisteme de realitate virtuală se definesc în mod diferit axele sistemului de referință universal față de modul defint mai sus. De exemplu, sistemul de referință universal poate fi definit cu axa x este îndreptată către Est, axa z către Sud, iar axa y în sus (spre cer). Un alt mod de interpretare care poate fi întâlnit în unele sisteme grafice, biblioteci sau programe de modelare este cu axa x către Nord, axa y către Est şi axa z îndreptată în jos (către pământ). Nu se poate spune că o convenție este mai bună decât alta, dar trebuie cunoscută convenția folosită şi efectuate transformările necesare între diferitele sisteme de referință care intervin în sistemul proiectat. În expunerea care urmează, se consideră un sistem de coordonate cu axa x către Sud şi axa y către Est, iar axa z reprezintă altitudinea (z).

Datele de teren pot fi obținute din hărți digitale (*Digital Terrain Elevation Data* – DTED) care memorează altitudinea într-o grilă spațială regulată, într-un sistem de coordonate geodezice. Distanța dintre puncte este dată în arc-secundă pe latitudine şi longitudine, cu valori care diferă în funcție de rezoluția datelor (1 arc-secundă sau 3 arc-secundă). La trecerea din coordonate geodetice în coordonate UTM, distanța între puncte nu se mai păstrează constantă, deci grila de puncte nu mai este regulată. Deoarece datele de aproximare a suprafeței terenului sunt altitudini cunoscute într-o grilă de puncte $p(x_i, y_i)$, orice linie paralelă cu axa z intersectează suprafața terenului într-un singur punct şi suprafața terenului poate fi construită pornind de la o funcție două variabile $z = F(x, y)$, definită într-un domeniu discret $\psi = \{p(x_i, y_i)\}$ [Cig97].

Construirea rețelei poligonale de aproximare a suprafeței astfel definite se poate face prin triangularizare în planul bidimensional, după care se adaugă altitudinea z_i a fiecărui punct. În acest mod se obține o suprafață poligonală

deschisă, care nu respectă condiţia de închidere a suprafeţelor de frontieră a poliedrelor, dar această condiţie nu este necesară în reprezentarea terenului.

În fig. 2.13 este reprezentată suprafaţa unei zone geografice de teren, obţinută prin triangularizarea tuturor punctelor grilei de altitudini din harta digitală corespunzătoare acelei zone.

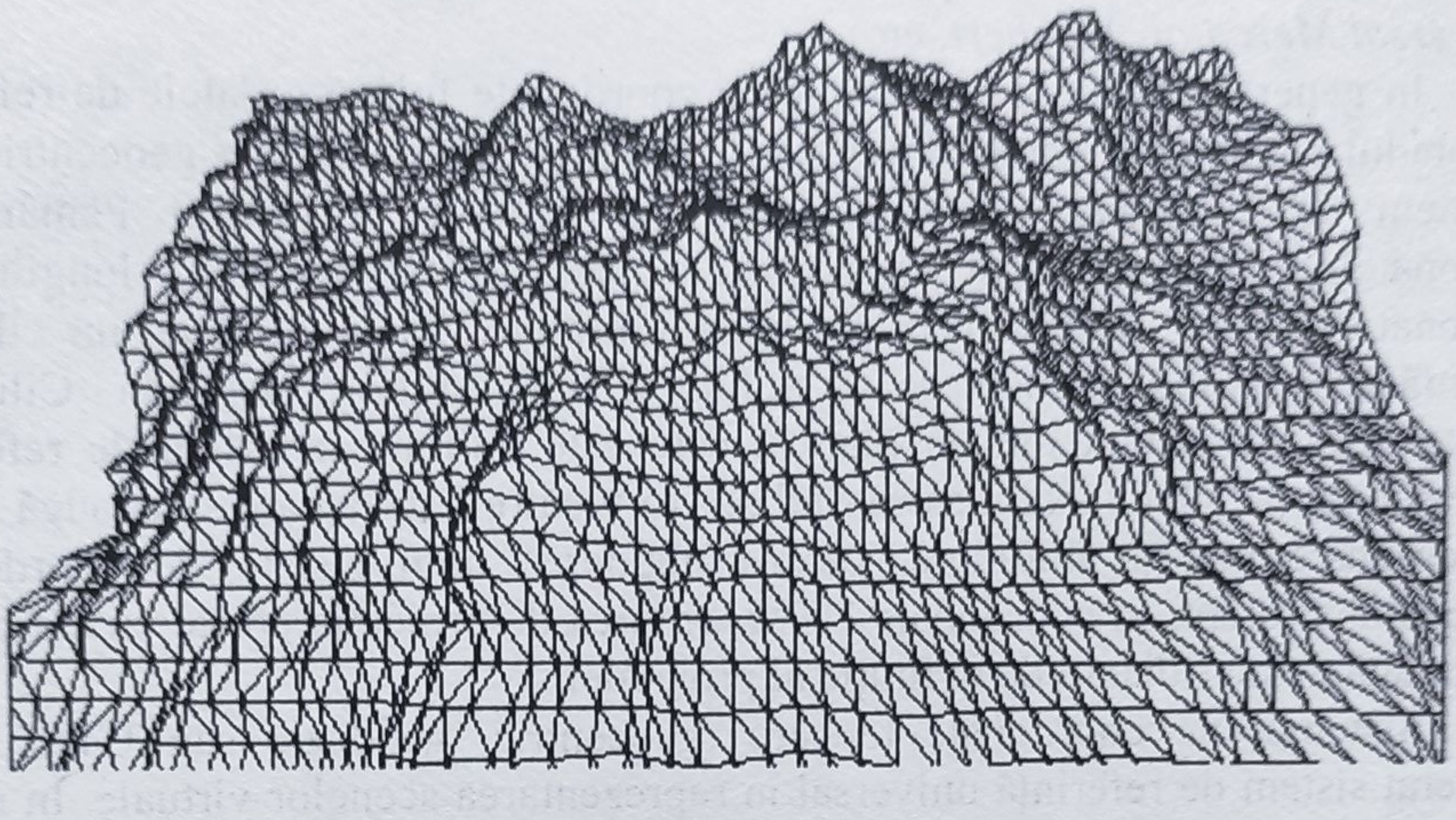


Fig. 2.13 Triangularizarea unei suprafeţe de teren.

Triangularizarea prin considerarea tuturor punctelor din domeniul dat creează o funcţie de aproximare F*(x, y) liniară pe porţiuni care aproximează suprafaţa dată prin funcţia F(x, y) cu eroare 0 în tot domeniul de definiţie. O astfel de soluţie este în general nesatisfăcătoare, datorită volumului mare de date rezultat. Terenul din fig. 2.14 a fost definit pe o grilă de 61×61 puncte cu rezoluţia de 3 arc-secunde şi au rezultat 7200 de triunghiuri, dat fiind că s-au inserat toate punctele în triangularizare. Costul de stocare şi prelucrare a acestor date poate fi inacceptabil, iar, pe de altă parte, se poate ca variaţiile locale mici să nu prezinte interes în aplicaţiile de realitate virtuală.

Alternativa o constitue construirea unei triangularizări care utilizează numai o parte din punctele domeniului de definiţie şi care respectă condiţia de precizie impusă, formulată astfel:

$$|F^*(x, y) - F(x, y)| \le \delta, \ \forall\ p(x,y) \in \psi \qquad (2.7)$$

unde δ reprezintă eroarea maximă impusă (fig. 2.14).

Una din metodele de construire a unui model care să respecte condiţia 2.7 este metoda selecţiei punctelor [Fjall93], care calculează o secvenţă $T_0, T_0, \ldots, T_m$ de triangularizări pentru submulţimi succesiv crescătoare ale mulţimii ψ a datelor de intrare. În fig. 2.14 s-au reprezentat proiecţiile în planul xy ale unor triangularizări prin metoda selecţiei punctelor. În triangularizarea iniţială T_0 se consideră punctele de pe frontiera domeniului şi se poate folosi orice metodă de triangularizare Delaunay în plan cunoscută (metoda cercului vid, metoda incrementală, etc.). După triangularizarea plană se adaugă coordonata z fiecărui punct din diagrama de triangularizare.

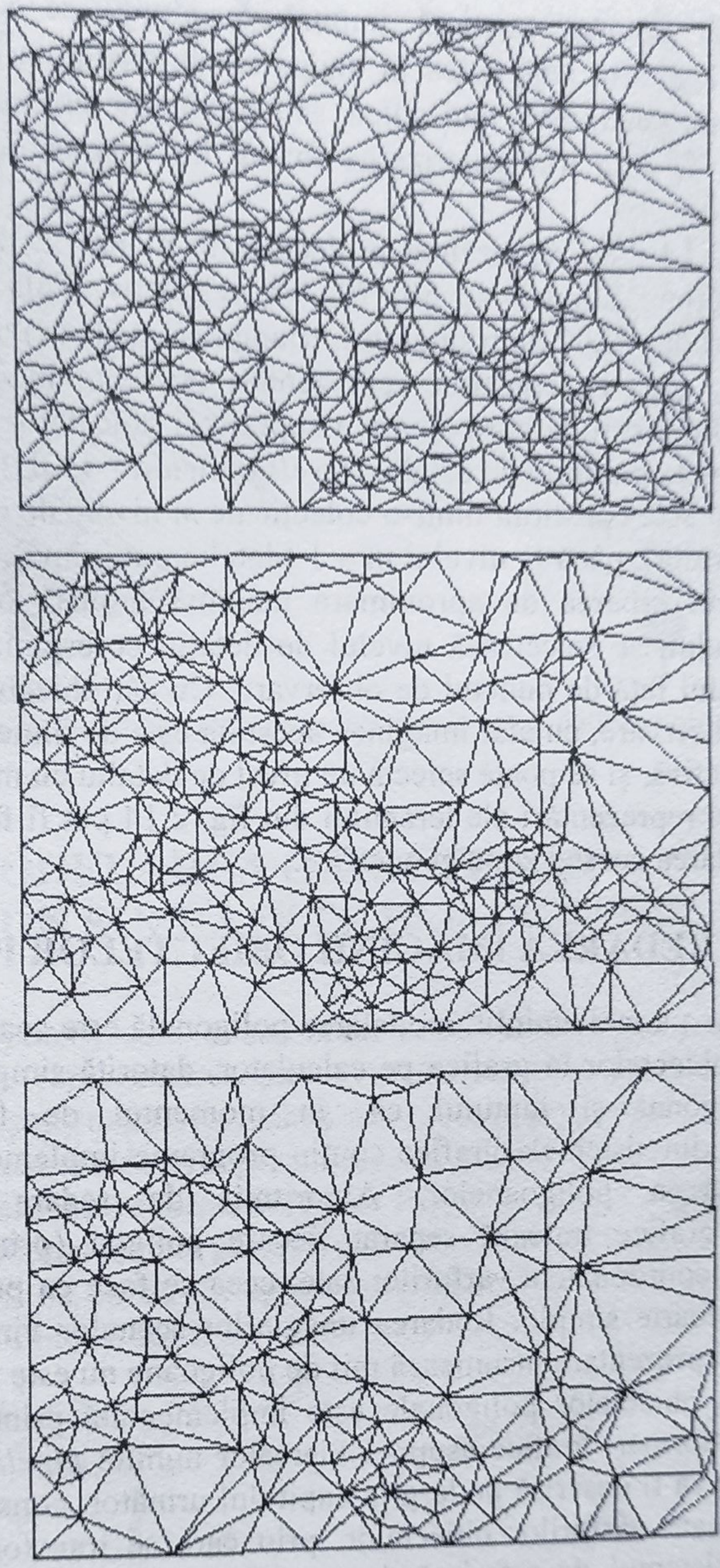


Fig. 2.14 Triangularizarea datelor de teren cu eroare maximă impusă.
(a) $\delta = 50$; (b) $\delta = 70$; (c) $\delta = 90$.

Fiecărei triangularizări T_k îi corespunde o funcție liniară pe porțiuni F_k, astfel că, în interiorul fiecărui triunghi al triangularizării T_k, F_k este dată de ecuația planului ce trece prin vârfurile triunghiului respectiv. Pentru fiecare triangularizare T_k există o locație $p_i(x_i, y_i)$ pentru care distanța la planul corespunzător funcției F_k (numită deviație) este maximă. Dacă în triangularizarea T_k deviația maximă este

mai mare decât eroarea maximă impusă δ, atunci se construieşte o nouă triangularizare T_{k+1} prin adăugarea în diagrama de triangularizare a punctului p_i. Dacă deviaţia maximă este mai mică sau egală cu eroarea maximă impusă, triangularizarea T_k este triangularizarea finală, iar funcţia $F_k = F^*(x, y)$ pentru eroarea impusă δ.

În fig. 2.14 s-au reprezentat proiecţiile în planul xy ale triangularizărilor având erori maxime impuse de 50, 70, 90, respectiv. Faţă de triangularizarea iniţială care a produs 7200 de triunghiuri, triangularizările (a), (b), (c) au 940, 512, 300 triunghiuri, ceea ce constitue reprezentări eficiente în condiţiile de eroare impuse. Controlul erorii de aproximare a suprafeţei obiectelor permite construirea modelelor cu nivele multiple de detaliu (*levels of detail* – LOD).

Modelul este constituit dintr-o colecţie de *m* nivele de detaliu, de la nivelul 0 (detaliere maximă), până la nivelul m – 1 (detaliere minimă). Un nivel de detaliu se defineşte prin eroarea de aproximare maximă impusă δ. În cursul redării imaginii obiectului, se selectează nivelul de detaliu corespunzător, în funcţie de distanţa obiectului faţă de punctul de observare. Cu cât obiectul este mai depărtat de punctul de observare, cu atât imaginea acestuia este de dimensiune mai mică în proiecţia perspectivă, şi se poate selecta un nivel de detaliu cu mai puţine elemente.

Cele trei reprezentări ale terenului din fig. 2.14 pot fi folosite ca nivele de detaliu în modelarea acestei zone geografice.

2.1.8 REDAREA IMAGINII OBIECTELOR POLIGONALE

Aşa cum s-a mai amintit, modelarea poligonală este cea mai folosită formă de modelare a obiectelor în grafica pe calculator, datorită simplităţii reprezentării modelului poligonal şi faptului că, în momentul de faţă, majoritatea acceleratoarelor din sistemele grafice conţin programe implementate hardware de redare eficientă a poligoanelor. Algoritmii de redare implementaţi în acceleratoarele grafice tratează separat fiecare poligon (primitivă geometrică) reprezentat prin coordonatele vârfurilor sale, ceea ce face ca procesul de redare a imaginii să fie foarte simplu. Redarea imaginilor scenelor virtuale compuse din obiecte a căror reprezentare însumează mii de poligoane nu este ceva neobişnuit.

Redarea obiectelor poligonale este implementată printr-o succesiune de operaţii de transformări grafice asupra obiectelor numită *pipeline grafic*. Această succesiune, care va fi descrisă pe larg în capitolul următor, constă din transformări geometrice aplicate vârfurilor obiectelor, prin care se transformă fiecare faţă a obiectului din sistemul de referinţă de modelare într-un sistem de referinţă de afişare, urmată de transformarea de redare, prin care se obţine culoare pixelilor care se afişează pe display. În fig. 2.15 este redată imaginea avionului F-16 modelat prin 2413 feţe poligonale şi redat prin prelucrarea fiecărei feţe, în modul cu feţe "pline" şi umbrire, şi în modul "wireframe". Pentru toate celelalte modele de reprezentare a obiectelor (reprezentarea prin reţele de petice, prin compunerea obiectelor sau prin divizarea spaţială), pe lângă redarea directă a modelului respectiv, există şi posibilitatea de redare prin deducerea mai întâi a reprezentării poligonale corespunzătoare, urmată de folosirea algoritmilor de redare poligonală.

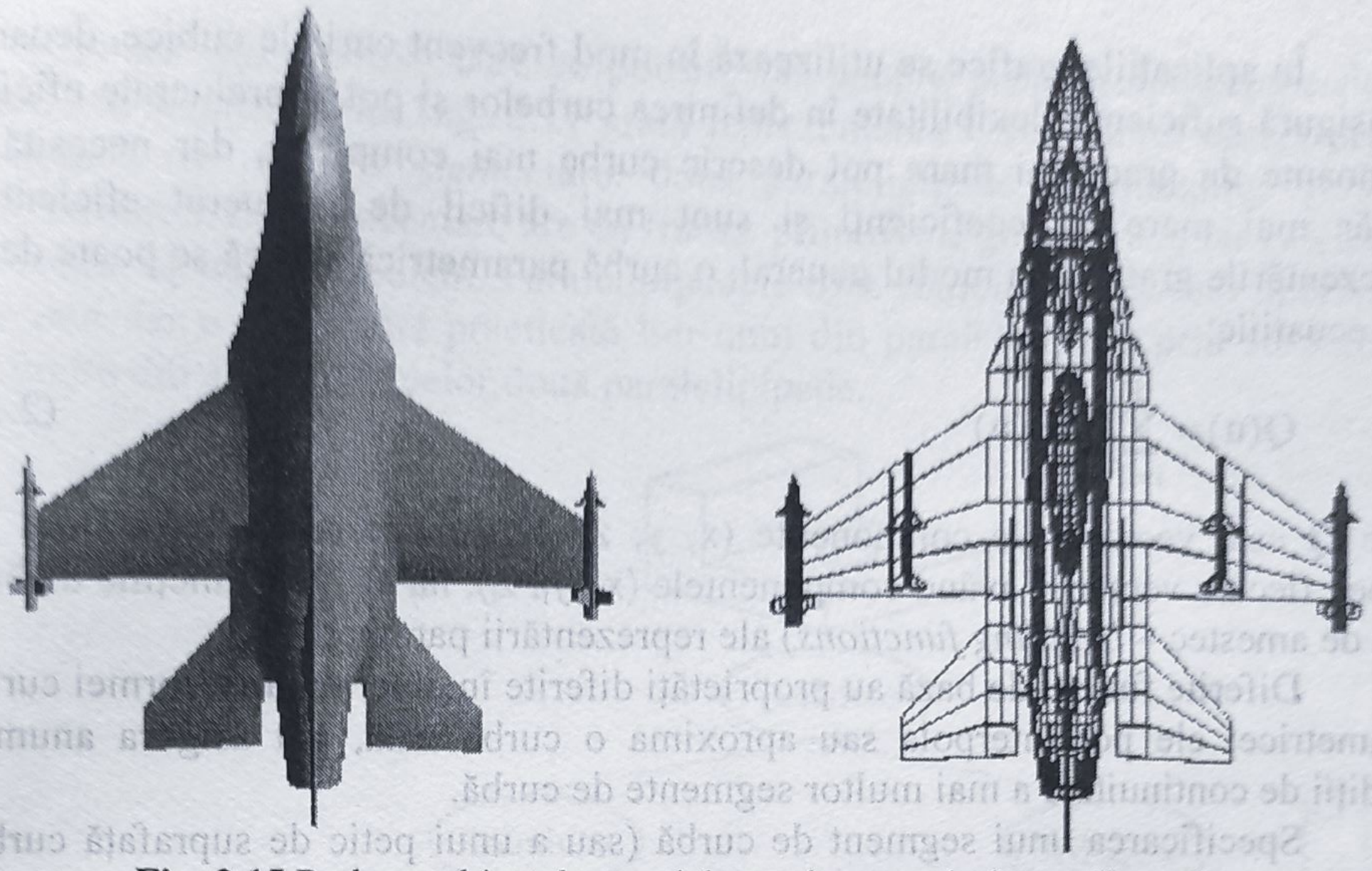


Fig. 2.15 Redarea obiectelor modelate prin rețea de fețe poligonale.

2.2 MODELAREA OBIECTELOR PRIN SUPRAFEȚE PARAMETRICE

Un petic (*patch*) este o suprafață curbă definită parametric în spațiul tridimensional. Prin reprezentarea parametrică, punctele de pe suprafață se pot calcula secvențial, pentru diferite valori ale parametrilor, mult mai simplu decât prin rezolvarea sistemului de ecuații care descriu implicit suprafața. Reprezentarea parametrică a curbelor şi a suprafețelor este folosită în proiectarea şi în modelarea obiectelor, pentru obținerea unei precizii mai ridicate de aproximare.

Un segment de curbă în spațiul tridimensional poate fi definit printr-un sistem de ecuații în funcție de un parametru:

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \tag{2.8}$$

Aceasta este o curbă cubică, definită ca formă şi mărime de cei 12 coeficienți, numiți coeficienți algebrici ai curbei. În notație vectorială se poate scrie forma parametrică a unei curbe cubice:

$$\mathbf{Q}(u) = \mathbf{A}u^3 + \mathbf{B}u^2 + \mathbf{C}u + \mathbf{D} \tag{2.9}$$

unde vectorul **Q** are componentele (x, y, z), iar vectorii **A**, **B**, **C**, **D** au componentele (a_x, a_y, a_z), (b_x, b_y, b_z), (c_x, c_y, c_z), (d_x, d_y, d_z), respectiv, iar u este cuprins în intervalul închis [0,1].

În aplicațiile grafice se utilizează în mod frecvent curbele cubice, deoarece ele asigură suficientă flexibilitate în definirea curbelor şi pot fi prelucrate eficient. Polinoame de grad mai mare pot descrie curbe mai complexe, dar necesită un număr mai mare de coeficienți şi sunt mai dificil de prelucrat eficient în reprezentările grafice. La modul general, o curbă parametrică cubică se poate defini prin ecuațiile:

$$\mathbf{Q}(u) = \sum_{i=0}^{3} \mathbf{P}_i \mathbf{B}_i(u) \qquad (2.10)$$

unde **Q** este vectorul de componente (x, y, z), $\mathbf{P}_i$ sunt punctele de control ale curbei, fiecare vector $\mathbf{P}_i$ având componentele (x_i, y_i, z_i), iar $\mathbf{B}_i$ sunt funcțiile de bază (sau de amestec – *blending functions*) ale reprezentării parametrice.

Diferite funcții de bază au proprietăți diferite în determinarea formei curbei parametrice: ele pot interpola sau aproxima o curbă dată, pot asigura anumite condiții de continuitate a mai multor segmente de curbă.

Specificarea unui segment de curbă (sau a unui petic de suprafață curbă) printr-un set de puncte de control este o metodă de bază în proiectarea grafică interactivă: proiectantul defineşte punctele de control; curba este generată şi vizualizată interactiv; dacă forma curbei nu este mulțumitoare, proiectantul modifică unul sau mai multe puncte de control, până obține rezultatul dorit.

Cele mai utilizate tipuri de curbe şi suprafețe parametrice în proiectarea grafică sunt curbele (şi suprafețele) Bézier şi B-spline. Modelarea curbelor şi a suprafețelor parametrice este prezentată detaliat în capitolul 8.

2.3 MODELAREA PRIN COMPUNEREA OBIECTELOR

Modelarea prin compunerea obiectelor (*Constructive Solid Geometry* – CSG) se foloseşte atunci când un obiect poate fi obținut prin combinarea mai multor obiecte elementare, numite primitive geometrice.

Primitivele geometrice utilizate sunt sfere, conuri, cilindri sau paralelipipede dreptunghice şi sunt combinate folosind operatori booleni şi transformări liniare. Un obiect complex este reprezentat printr-un arbore, ale cărui frunze sunt primitivele geometrice, iar nodurile memorează operatori booleeni sau transformări liniare. În fig. 2.16 este prezentată operația de reuniune a două obiecte elementare.

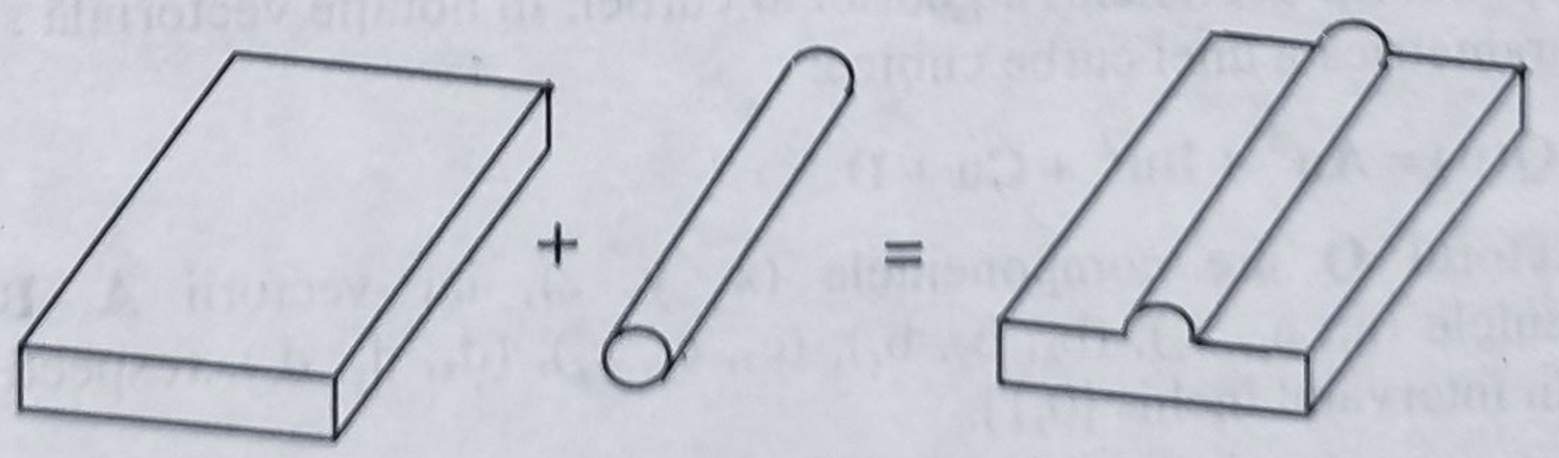


Fig. 2.16 Operația de reuniune a două primitive geometrice.

Alte operaţii posibile care se pot efectua asupra primitivelor geometrice sunt scăderea şi intersecţia. Fig. 2.17 arată reprezentarea CSG a unui obiect prin combinarea a trei obiecte elementare: două paralelipipede dreptunghice şi un cilindru. Arborele de reprezentare are ca frunze primitivele geometrice, iar celelate noduri conţin operatorii booleni. Paralelipipedele sunt combinate folosind operaţia de reunuine, iar o gaură este practicată într-unul din paralelipipede prin scăderea unui cilindru din ansamblul celor două paralelipipede.

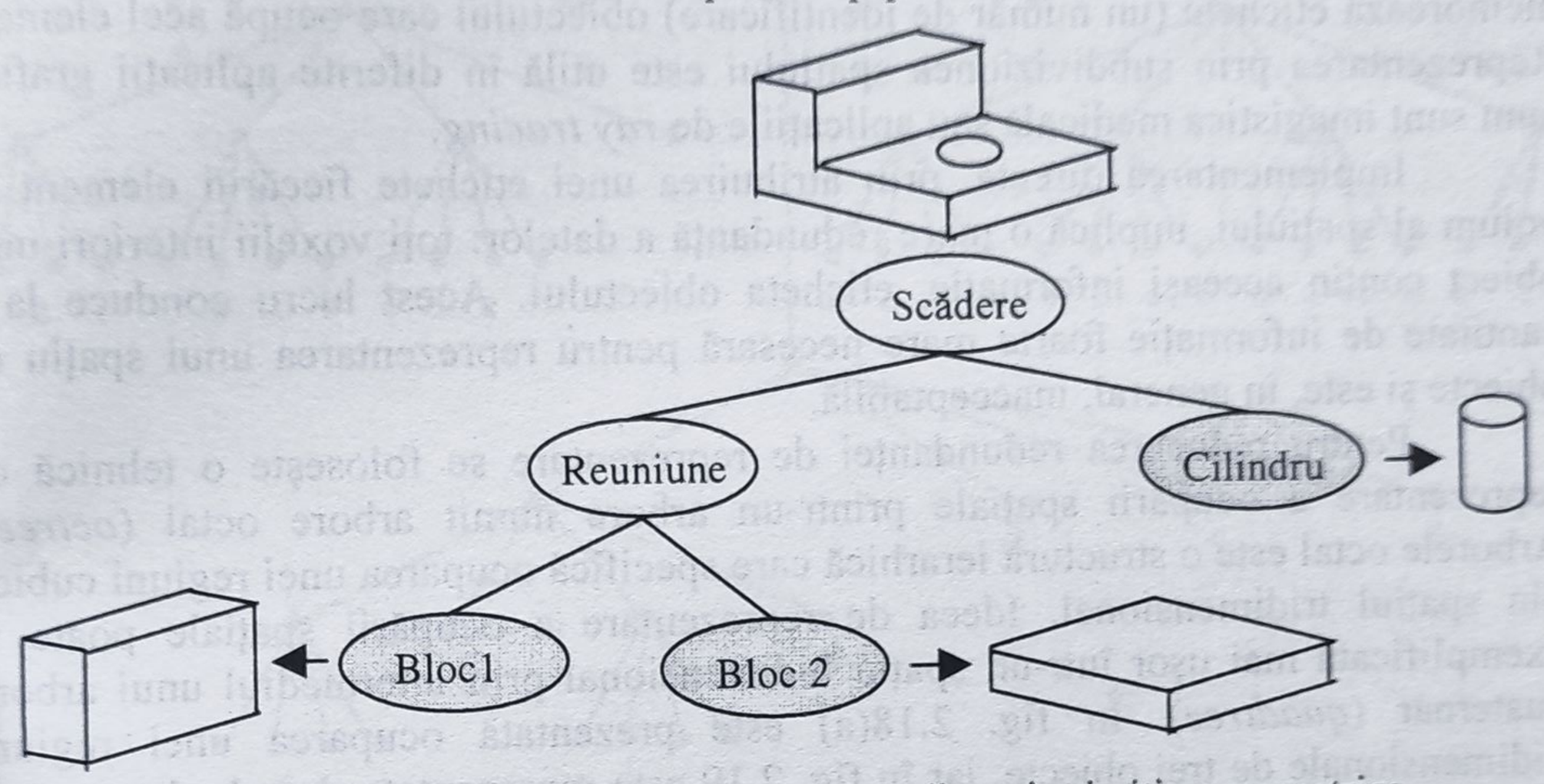


Fig. 2.17 Arborele de construire a unui obiect din trei primitive geometrice.

Deoarece arborele de reprezentare memorează atât operaţiile boolene, cât şi forma primitivelor geometrice, operaţiile de modificare ale obiectului compus sunt relativ simple. De exemplu, o gaură într-un obiect se modifică prin modificarea poziţiei sau a dimensiunii primitivei geometrice folosite în operaţia de scădere. O astfel de modificare este mult mai dificilă în reprezentarea prin suprafaţa de frontieră a modelelor.

Redarea imaginii obiectelor CSG se poate face prin mai multe metode:

- Redarea directă a obiectului în aplicaţiile de *ray-tracing*.
- Conversia modelului CSG în reprezentare prin suprafaţa de frontieră (B-rep) şi aplicarea procedurilor standard de redare a poligoanelor.
- Conversia modelului CSG în reprezentare prin divizarea spaţiului (în elemente numite voxeli) şi redarea volumului corespunzător

Tehnica ray-tracing generează imaginea obiectelor prin intersecţia acestora cu raze de proiecţie şi permite obţinerea unor efecte deosebit de realiste în modelarea reflexiei, transparenţei şi a umbrelor.

Un dezavantaj important al modelării CSG este acela că nu orice fel de obiect poate fi obţinut prin combinarea unor primitive geometrice simple. De exemplu, modelul unei statui (folosit în crearea muzeelor virtuale) sau modelul unui organ anatomic (folosit în simulările de intervenţii chirurgicale) nu poate fi obţinut prin combinarea unor primitive geometrice simple.

2.4 MODELAREA PRIN DIVIZARE SPAȚIALĂ

În tehnica de divizare spațială, se atribuie fiecărei subdiviziuni a spațiului tridimensional câte o etichetă în funcție de obiectul care ocupă acea subdiviziune.

În acestă tehnică se consideră spațiul tridimensional compus dintr-un număr de m×n×k volume elementare (numite *voxeli*) şi pentru fiecare voxel se memorează eticheta (un număr de identificare) obiectului care ocupă acel element. Reprezentarea prin subdiviziunea spațiului este utilă în diferite aplicații grafice, cum sunt imagistica medicală sau aplicațiile de *ray tracing*.

Implementarea directă, prin atribuirea unei etichete fiecărui element de volum al spațiului, implică o mare redundanță a datelor: toți voxelii interiori unui obiect conțin aceeaşi informație, eticheta obiectului. Acest lucru conduce la o cantitate de informație foarte mare necesară pentru reprezentarea unui spațiu de obiecte şi este, în general, inacceptabilă.

Pentru reducerea redundanței de reprezentare se foloseşte o tehnică de reprezentare a ocupării spațiale printr-un arbore numit arbore octal (*octree*). Arborele octal este o structură ierarhică care specifică ocuparea unei regiuni cubice din spațiul tridimensional. Ideea de reprezentare a ocupării spațiale poate fi exemplificată mai uşor într-un spațiu bidimensional prin intermediul unui arbore cuaternar (*quadtree*). În fig. 2.18(a) este prezentată ocuparea unei regiuni bidimensionale de trei obiecte, iar în fig. 2.19 este reprezentat arborele de ocupare a regiunii.

Arborele se creează pornind cu o regiune pătrată în plan, constituind întreaga zonă care se modelează şi care este reprezentată prin nodul rădăcină al arborelui cuaternar. În cazul spațiului tridimensional, acestă regiune este un cub. Fiecare regiune, începând cu regiunea inițială, se subdivide în patru subregiuni, reprezentate ca patru noduri fii în arbore. În fig. 2.18(b) se arată ordinea de numerotare a nodurilor fii obținuți prin divizarea unei subregiuni.

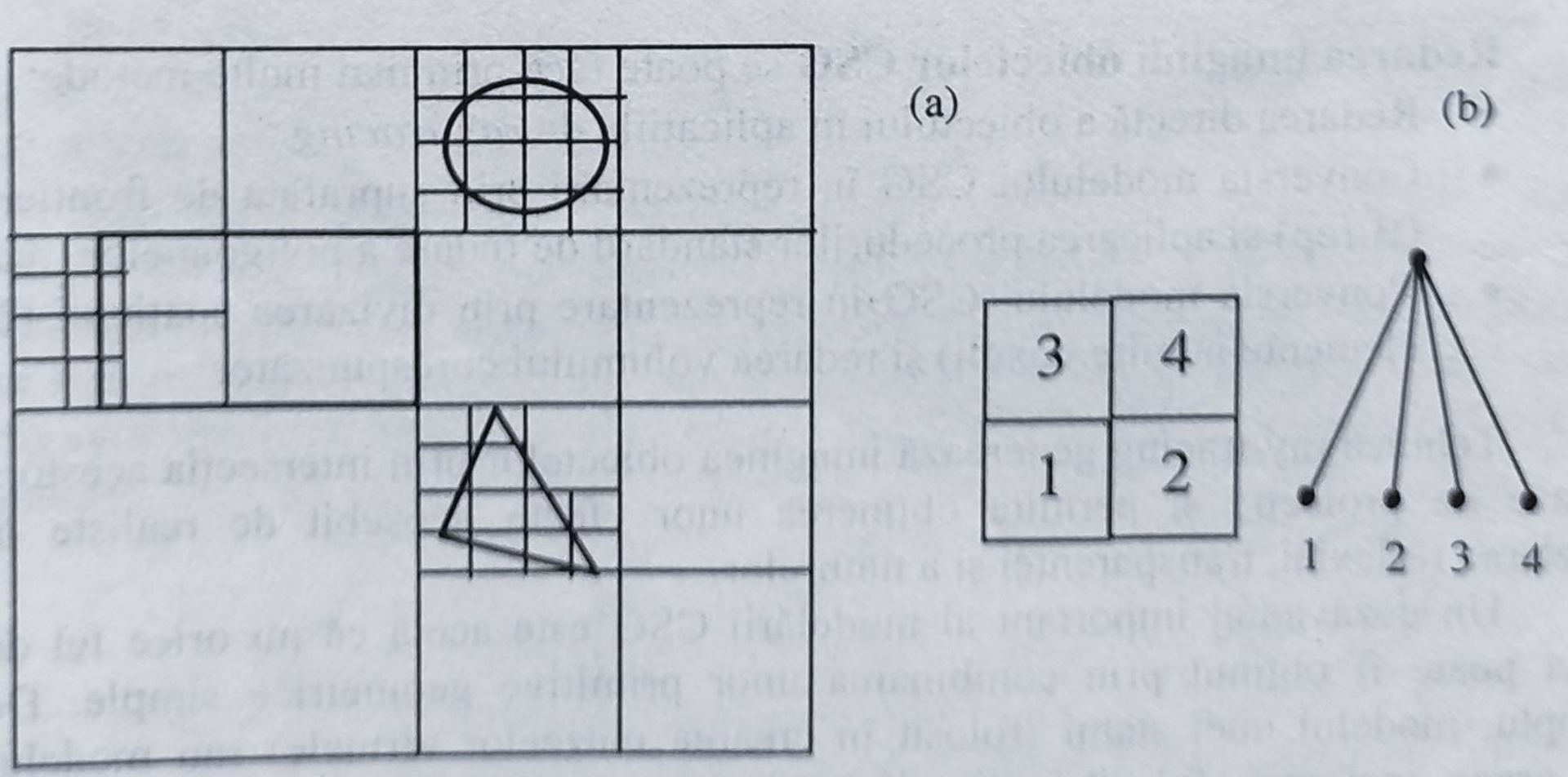


Fig. 2.18 (a) Ocuparea unei regiuni plane.
(b) Ordinea de numerotare a nodurilor fii.

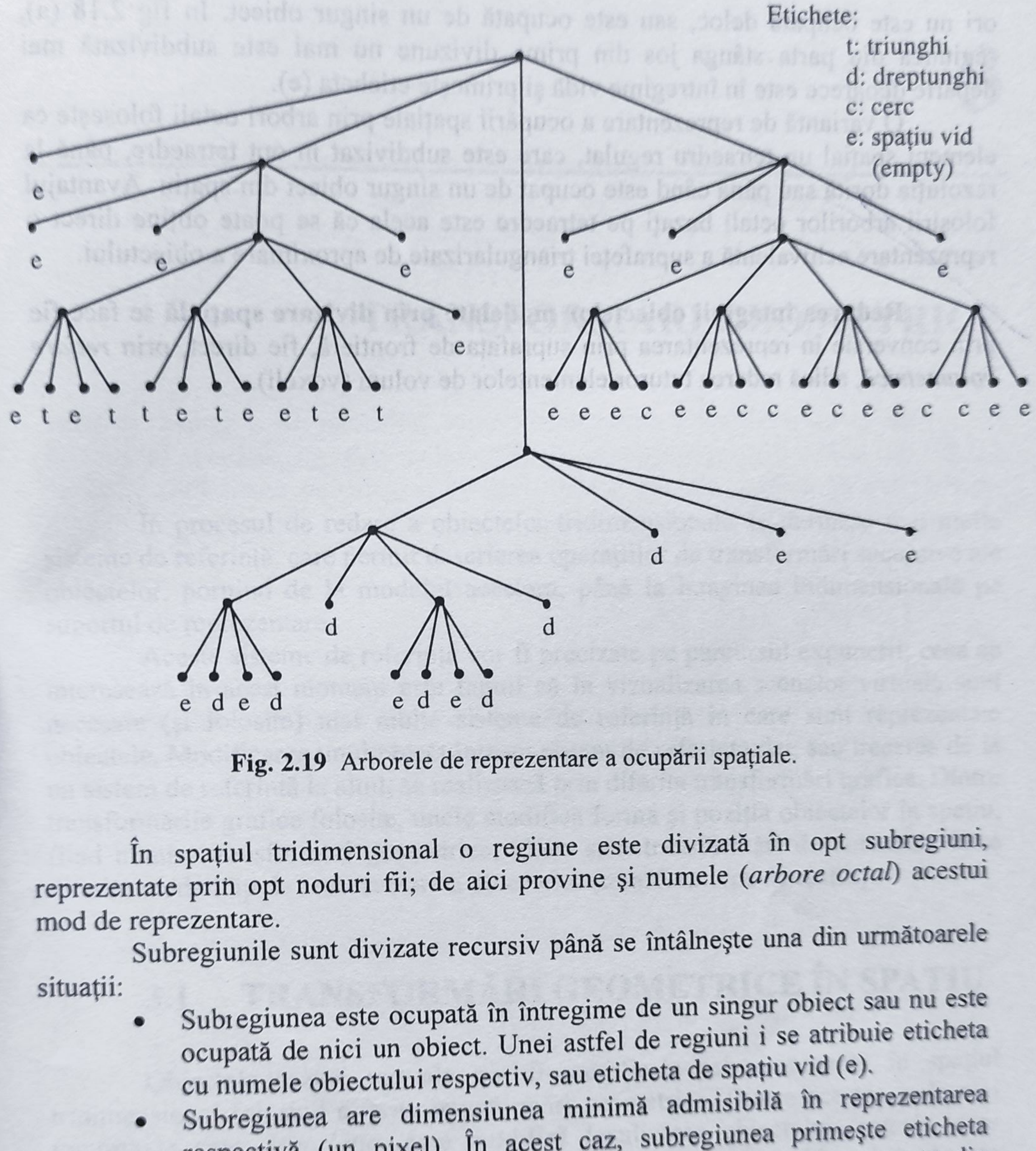


Fig. 2.19 Arborele de reprezentare a ocupării spațiale.

În spațiul tridimensional o regiune este divizată în opt subregiuni, reprezentate prin opt noduri fii; de aici provine şi numele (*arbore octal*) acestui mod de reprezentare.

Subregiunile sunt divizate recursiv până se întâlneşte una din următoarele situații:

- Subregiunea este ocupată în întregime de un singur obiect sau nu este ocupată de nici un obiect. Unei astfel de regiuni i se atribuie eticheta cu numele obiectului respectiv, sau eticheta de spațiu vid (e).
- Subregiunea are dimensiunea minimă admisibilă în reprezentarea respectivă (un pixel). În acest caz, subregiunea primeşte eticheta obiectului (sau a spațiului vid) care ocupă cea mai mare parte din subregiunea respectivă.

În arborele de reprezentare a ocupării spațiului tridimensional, dimensiunea minimă până la care se divizează subregiunile este dimensiunea unui voxel. În arborele de reprezentare sunt două tipuri de noduri frunză: noduri care au o etichetă a unui obiect sau noduri care au eticheta spațiului vid. Compactarea volumului de date în această reprezentare față de reprezentarea prin etichetarea fiecărui element (pixel sau voxel) provine din faptul că se poate ca o întreagă subregiune (mai mare decât un voxel) să primească o singură etichetă, dacă este

ori nu este ocupată deloc, sau este ocupată de un singur obiect. În fig 2.18 (a), regiunea din parta stânga jos din prima divizune nu mai este subdivizată mai departe deoarece este în întregime vidă şi primeşte eticheta (e).

O variantă de reprezentare a ocupării spațiale prin arbori octali foloseşte ca element spațial un tetraedru regulat, care este subdivizat în opt tetraedre, până la rezoluția dorită sau până când este ocupat de un singur obiect din spațiu. Avantajul folosirii arborilor octali bazați pe tetraedre este acela că se poate obține direct o reprezentare echivalentă a suprafeței triangularizate de aproximare a obiectului.

Redarea imaginii obiectelor modelate prin divizare spațială se face fie prin conversie în reprezentarea prin suprafața de frontieră, fie direct, prin *redare volumetrică*, adică redarea tuturor elementelor de volum (voxeli).

TRANSFORMĂRI GEOMETRICE

În procesul de redare a obiectelor tridimensionale se definesc mai multe sisteme de referință, care permit descrierea operațiilor de transformări succesive ale obiectelor, pornind de la modelul acestora, până la imaginea bidimensională pe suportul de reprezentare.

Aceste sisteme de referință vor fi precizate pe parcursul expunerii; ceea ce interesează în acest moment este faptul că în vizualizarea scenelor virtuale sunt necesare (şi folosite) mai multe sisteme de referință în care sunt reprezentate obiectele. Modificarea unui obiect într-un sistem de referință dat, sau trecerea de la un sistem de referință la altul, se realizează prin diferite transformări grafice. Dintre transformările grafice folosite, unele modifică forma şi poziția obiectelor în spațiu, fiind numite transformări geometrice, altele sunt transformări de conversie între diferite modalități de reprezentare a obiectelor (transformarea de rastru).

3.1 TRANSFORMĂRI GEOMETRICE ÎN SPAȚIU

Obiectele scenei virtuale pot fi modificate sau manevrate în spațiul tridimensional folosind diferite transformări geometrice. Dintre acestea, cele mai importante sunt: *translația*, care modifică localizarea obiectului; *rotația*, care modifică orientarea; *scalarea*, care modifică dimensiunea obiectului. Aceste transformări sunt denumite transformări geometrice primitive.

3.1.1 TRANSFORMĂRI GEOMETRICE PRIMITIVE

Translația este transformarea prin care toate punctele se deplasează în aceeaşi direcție şi cu aceeaşi distanță între punct şi transformatul său. Translația se poate descrie printr-un vector de translație **T**, având componentele t_x, t_y, t_z pe cele trei axe de coordonate; un punct P(x,y,z) se transformă în punctul P'(x',y',z') astfel:

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases}$$

În notație matriceală, transformarea prin translație cu vectorul de translație **T**, având componentele t_x, t_y, t_z pe cele trei axe de coordonate a unui punct P(x, y, z) în punctul P'(x', y', z') se exprimă printr-o însumare de matrice:

$$\mathbf{P'} = \mathbf{P} + \mathbf{T}, \text{ unde } \mathbf{T} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}; \text{ deci: } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Scalarea modifică coordonatele tuturor punctelor unui obiect folosind factorii de scală s_x, s_y, respectiv s_z pe cele trei axe de coordonate. În această transformare de scalare, numită scalare față de origine, un punct P(x, y, z) se transformă în punctul P'(x', y', z'), unde:

$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \\ z' = z \cdot s_z \end{cases}$$

Pentru scrierea sub formă matriceală a acestor relații de transformare, se definește matricea de scalare **S** de dimensiune 3×3 astfel:

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Rezultă relația de transformare de scalare în notație matriceală:

$$\mathbf{P'} = \mathbf{S}\,\mathbf{P}, \text{ adică } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotația cu un unghi θ în raport cu o axă D este o transformare prin care orice punct P care nu este situat pe dreapta D se transformă într-un punct P' astfel încât P și P' sunt situate într-un plan perpendicular π pe dreapta D, unghiul PIP' este egal cu θ, iar modulele |IP| și |IP'| sunt egale (punctul I este intersecția dintre dreapta D și planul π). Prin această transformare toate punctele dreptei D sunt fixe și numai ele sunt puncte invariante ale transformării. Transformarea de rotație în raport cu o axă oarecare se descompune într-o succesiune de maximum trei transformări de rotație în raport cu axele de coordonate ale sistemului de referință.

Rotația în raport cu axa z cu un unghi θ transformă un punct P(x,y,z) în punctul P'(x',y',z'), ambele aflate în planul π perpendicular pe axa z. Pentru

deducerea relaţiilor de transformare se exprimă coordonatele punctelor P şi P' în acest plan în coordonate polare (fig. 3.1).

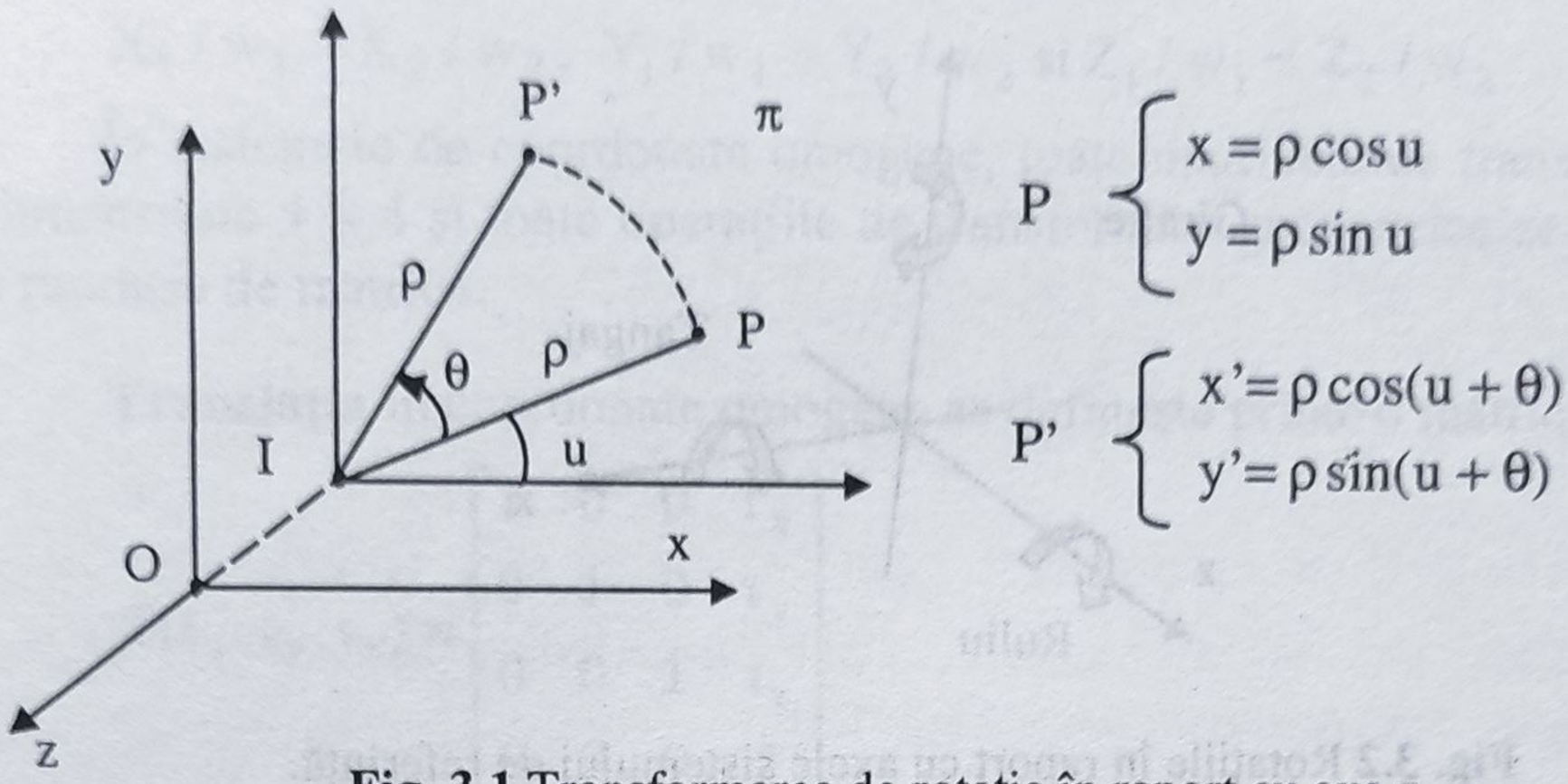


Fig. 3.1 Transformarea de rotaţie în raport cu axa z.

Se dezvoltă cosinusul şi sinusul sumei de unghiuri şi se obţin relaţiile:

$$x' = \rho(\cos u \cos\theta - \sin u \sin\theta) = x\cos\theta - y\sin\theta$$
$$y' = \rho(\sin u \cos\theta + \sin\theta \cos u) = x\sin\theta + y\cos\theta$$

Această transformare se poate scrie sub formă matriceală dacă se defineşte matricea de rotaţie $\mathbf{R_Z}(\theta)$ de dimensiune 3×3 astfel:

$$\mathbf{R_Z}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rezultă relaţiile de transformare de rotaţie a unui punct în raport cu axa z cu un unghi θ scrise sub formă matriceală:

$$\mathbf{P'} = \mathbf{R_Z P}, \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Pentru rotaţiile elementare ale unui punct în raport cu axele x şi y ale sistemului de referinţă se urmăreşte un raţionament asemănător şi se deduc relaţiile de transformare corespunzătoare.

Rotaţiile în raport cu axele de coordonate ale sistemului de referinţă sunt denumite tangaj (*pitch*), giraţie (*yaw*, *heading*) şi, respectiv, ruliu (*roll*). Această asignare depinde de convenţiile de definire a sistemului de referinţă universal. Pentru convenţia definită mai sus, tangajul este o rotaţie în raport cu axa x, giraţia este o rotaţie în raport cu axa y, iar ruliul este o rotaţie în raport cu axa z (fig. 3.2).

Aceste denumiri au originea în modul în care sunt definite mişcările unui avion poziţionat în spaţiu cu axa longitudinală orientată către z pozitiv: tangajul este rotaţia într-un plan vertical, care înclină botul avionului; giraţia este o mişcare

într-un plan orizontal, care schimbă direcția axei avionului, iar ruliul este rotația într-un plan vertical, care înclină aripile avionului.

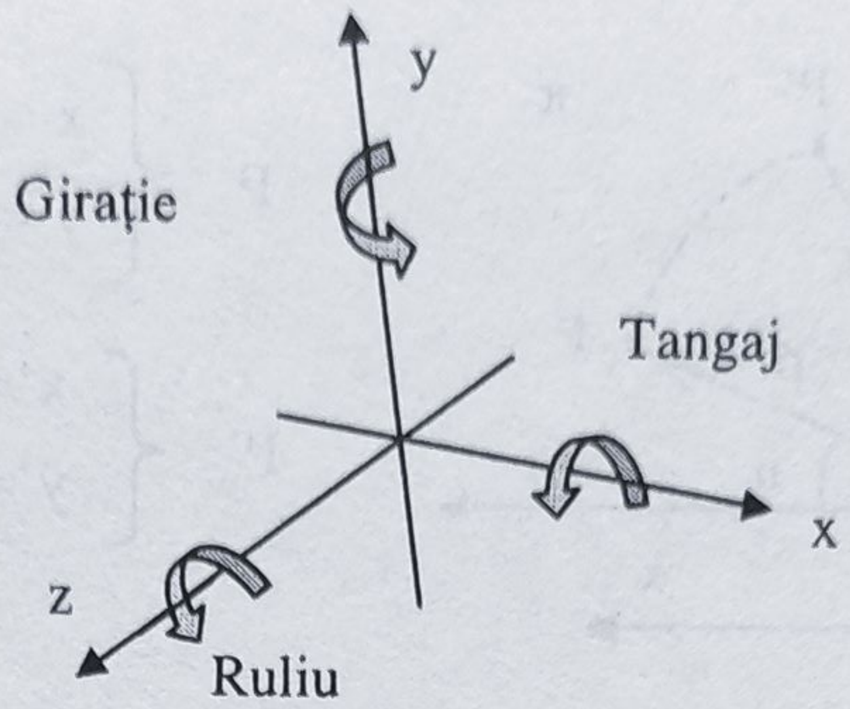


Fig. 3.2 Rotațiile în raport cu axele sistemului de referință.

Semnul rotațiilor în raport cu axele de coordonate se alege, prin convenție, după regula mâinii drepte sau după regula mâinii stângi. Dacă se cuprinde axa corespunzătoare cu patru degete ale mâinii (drepte, respectiv, stângi) astfel ca degetul mare să fie îndreptat în sensul pozitiv al axei, direcția celor patru degete indică sensul pozitiv al rotației. În această lucrare se adoptă regula mâinii drepte pentru sensul de rotație în raport cu axele de coordonate (fig. 3.2).

S-au obținut relațiile pentru transformările spațiale elementare (primitive), translația, scalarea față de originea sistemului de coordonate şi rotația în raport cu axele sistemului de coordonate. Tratarea într-un mod unitar a acestor transformări se poate face prin creşterea dimensionalității sistemului de coordonate cartezian, definindu-se un sistem de coordoonate cu 4 dimensiuni, numit sistem de coordonate omogene.

3.1.2 SISTEME DE COORDONATE OMOGENE

Se observă că transformările de scalare şi de rotație se reprezintă prin înmulțiri de matrice, iar translația se reprezintă prin însumare de matrice. Reprezentarea unitară şi combinarea transformărilor geometrice spațiale se poate face într-un sistem de coordonate cu patru dimensiuni, numit sistem de coordonate omogene. Un punct P(x,y,z) se reprezintă în sistemul de coordonate omogene ca P(X,Y,Z,w), unde $X = x \cdot w$, $Y = y \cdot w$, $Z = z \cdot w$, pentru orice factor de scară $w \neq 0$.

În general, un punct P(x, y, z) în sistemul de coordonate cartezian se transformă în sistemul de coordonate omogene prin alegerea $w = 1$, deci are coordonatele omogene P(x,y,z,1). Pentru transformarea inversă, se calculează coordonatele carteziene ale unui punct P(X,Y,Z,w), reprezentat în sistemul de coordonate omogene printr-o împărțire cu factorul de scară w astfel:

$$\begin{cases} x = X / w \\ y = Y / w \\ z = Z / w \end{cases} \tag{3.1}$$

În coordonate omogene, două puncte $P_1(X_1, Y_1, Z_1, w_1)$ şi $P_2(X_2, Y_2, Y_2, w_2)$ sunt egale dacă :

$$X_1 / w_1 = X_2 / w_2,\ Y_1 / w_1 = Y_2 / w_2 \text{ si } Z_1 / w_1 = Z_2 / w_2$$

În sistemele de coordonate omogene, toate matricele de transformări sunt de dimensiune 4×4 şi toate operaţiile de transformări geometrice se pot exprima prin produse de matrice.

Translaţia în coordonate omogene se defineşte printr-o matrice:

$$\mathbf{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

Transformarea de translaţie a unui punct $P(X, Y, Z, w)$ reprezentat în coordonate omogene, în punctul $P'(X', Y', Z', w')$ se exprimă ca un produs de matrice:

$$\mathbf{P'} = \mathbf{T}\,\mathbf{P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă: } \begin{cases} X' = X + wt_x \\ Y' = Y + wt_y \\ Z' = Z + wt_z \\ w' = w \end{cases} \tag{3.3}$$

Teoretic, pentru a se obţine coordonatele cartezine ale punctului transformat P', trebuie să se execute operaţia de împărţire cu factorul de scară w' pentru fiecare coordonată:

$$\begin{cases} x' = X' / w' \\ y' = Y' / w' \\ z' = Z' / w' \end{cases}$$

Dar, dacă se alege $w = 1$, rezultă $w' = 1$ şi împărţirea nu mai este necesară. În general, transformările geometrice primitive conservă valoarea factorului de scară şi, dacă se alege $w = 1$, împărţirea cu w' (numită împărţirea omogenă) nu este necesară.

Scalarea faţă de origine se reprezintă în sistemul de coordonate omogene prin matricea:

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Transformarea de scalare a unui punct reprezentat în coordonate omogene $P(X, Y, Z, w)$, în punctul $P'(X', Y', Z', w')$ este dată de relaţiile:

$$\mathbf{P'} = \mathbf{SP}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă: } \begin{cases} X' = s_x X \\ Y' = s_y Y \\ Z' = s_z Z \\ w' = w \end{cases} \tag{3.5}$$

Dacă factorii de scalare sunt egali ($s_x = s_y = s_z$), scalarea se numeşte uniformă, şi păstrează forma obiectului. Dacă factorii de scalare diferă, obiectul este deformat, iar scalarea se numeşte neuniformă.

Transformările de rotaţie în raport cu axele sistemului de referinţă exprimate în coordonate omogene sunt următoarele:

Rotaţia în raport cu axa x (tangaj) cu un unghi θ :

$$\mathbf{R_X}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

În această transformare, axa x rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R_X}(\theta)$:

$$\mathbf{P'} = \mathbf{R_X P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă } \begin{cases} X' = X \\ Y' = Y\cos\theta - Z\sin\theta \\ Z' = Y\sin\theta + Z\cos\theta \\ w' = w \end{cases} \tag{3.7}$$

Rotaţia în raport cu axa y (giraţie) cu un unghi χ :

$$\mathbf{R_Y}(\chi) = \begin{bmatrix} \cos\chi & 0 & \sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

În această transformare, axa y rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R_Y}(\chi)$:

$$\mathbf{P'} = \mathbf{R_Y P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\chi & 0 & \sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă: } \begin{cases} X' = X\cos\chi + Z\sin\chi \\ Y' = Y \\ Z' = -X\sin\chi + Z\cos\chi \\ w' = w \end{cases} \quad (3.9)$$

Rotaţia în raport cu axa z (ruliu), cu un unghi ρ :

$$\mathbf{R}_Z(\rho) = \begin{bmatrix} \cos\rho & -\sin\rho & 0 & 0 \\ \sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

În această transformare, axa z rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R}_Z(\rho)$:

$$\mathbf{P'} = \mathbf{R}_Z\mathbf{P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\rho & -\sin\rho & 0 & 0 \\ \sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă: } \begin{cases} X' = X\cos\rho - Y\sin\rho \\ Y' = X\sin\rho + Y\cos\rho \\ Z' = Z \\ w' = w \end{cases} \quad (3.11)$$

Toate matricele de transformare de rotaţie sunt matrice ortogonale şi ortonormate. Transformările geometrice elementare sunt transformări liniare, prin care liniile drepte şi suprafeţele plane sunt transformate în linii dreapte, respectiv suprafeţe plane. Din această cauză, pentru transformarea unui obiect tridimensional este suficient să se transforme toate vârfurile acestuia şi să se păstreze relaţiile topologice între vârfurile transformate, aceleaşi cu cele între vârfurile iniţiale.

Transformările mai complexe ale obiectelor în spaţiu se pot defini prin compunerea mai multor transformări primitive.

3.1.3 COMPUNEREA TRANSFORMĂRILOR GEOMETRICE

Compunerea mai multor tranformări elementare pentru obţinerea unei transformări complexe se obţine prin executarea succesivă a produsului fiecăriei matrice de transformare cu matricea de reprezentare a punctului iniţial sau rezultat dintr-o transformare precedentă. De exemplu, în fig. 3.3 este prezentată o transformare compusă a unui obiect.

Obiectul iniţial este un cub cu latura de două unităţi, amplasat cu centrul său în centrul sistemului de referinţă şi laturile orientate în direcţiile axelor de coordonate. După o scalare cu factorii de scară 2, 2, 2, o rotaţie cu un unghi de 30 grade în raport cu axa z şi o translaţie cu un vector de translaţie cu componente 8,0,0, se obţine un nou obiect cub, definit în acelaşi sistem de referinţă, dar cu alte dimensiuni şi localizare. Reprezentarea din fig. 3.3 conţine şi o transformare de

proiecţie perspectivă, pentru percepţia adâncimii (a distanţei faţă de observator a obiectelor), care va fi explicată ulterior. Una din feţele cubului (cea mai apropiată de observator) este desenată ca suprafaţă de culoare gri; celelalte feţe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare numită "cadru de sârmă" – *wireframe*).

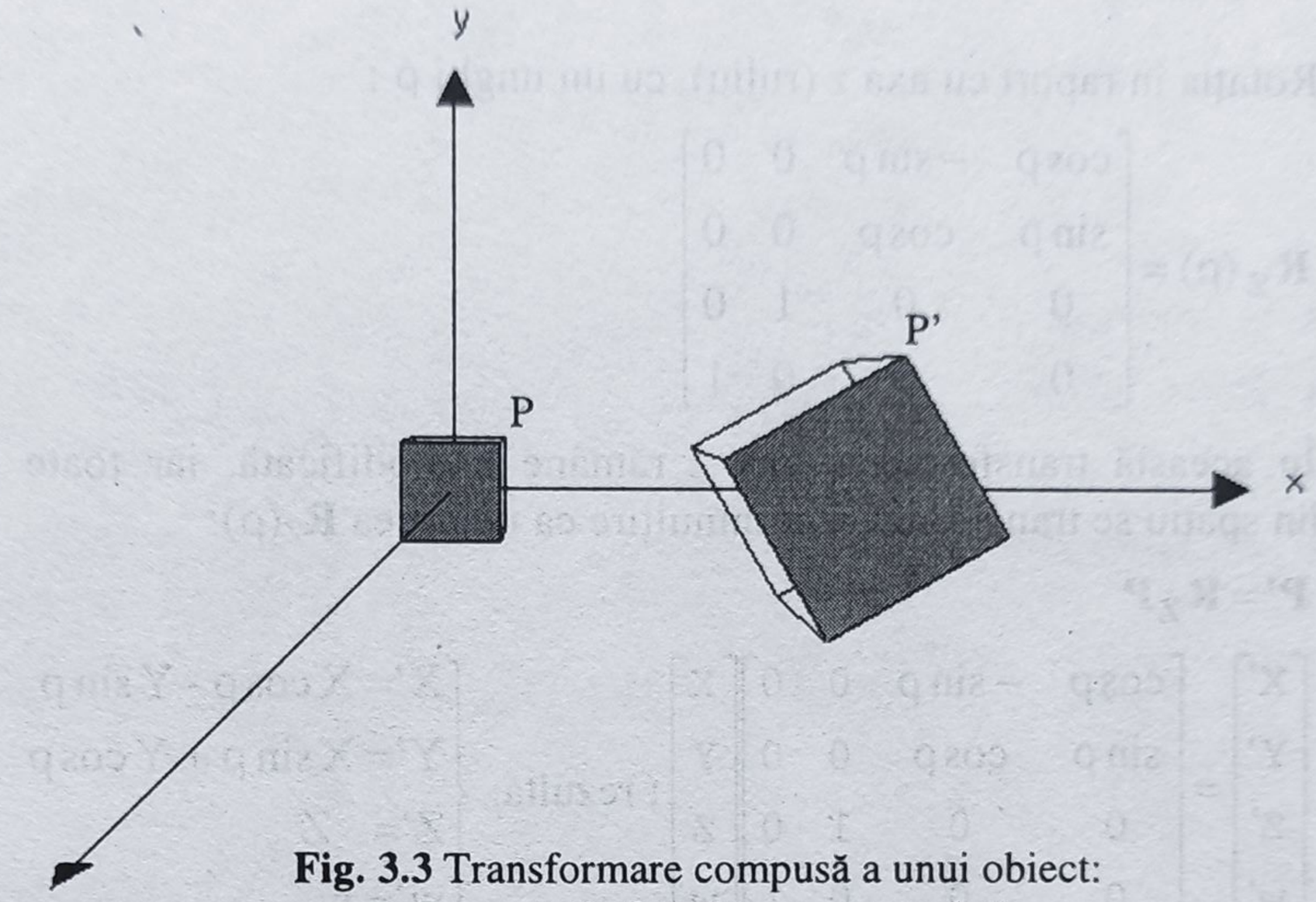


Fig. 3.3 Transformare compusă a unui obiect: scalare, rotaţie faţă de axa z, translaţie.

Transformarea efectuată asupra cubului din figura de mai sus se obţine prin aplicarea succesivă a trei transformări geometrice elementare (scalare, rotaţie şi translaţie) asupra fiecărui punct (vârf) al cubului. Pentru un vârf al cubului, reprezentat prin matricea coloană **P**, succesiunea de transformări este:

Scalarea: $\mathbf{P}_1 = \mathbf{S}\,\mathbf{P}$

Rotaţia în raport cu axa z: $\mathbf{P}_2 = \mathbf{R}_Z\,\mathbf{P}_1 = \mathbf{R}_Z(\mathbf{S}\,\mathbf{P})$

Translaţia: $\mathbf{P}' = \mathbf{T}\,\mathbf{P}_2 = \mathbf{T}(\mathbf{R}_Z(\mathbf{S}\,\mathbf{P})) = (\mathbf{T}\mathbf{R}_Z\mathbf{S})\,\mathbf{P}$

Nu este necesar să fie executate pe rând operaţiile de înmulţire cu matricele de transformare **S**, $\mathbf{R}_Z$, şi **T**, ci se poate calcula o matrice compusă a transformării **M**, care se aplică apoi fiecărui punct **P** al obiectului:

$$\mathbf{P}' = \mathbf{T}\mathbf{R}_Z\mathbf{S}\,\mathbf{P} = \mathbf{M}\,\mathbf{P}\text{, unde } \mathbf{M} = \mathbf{T}\mathbf{R}_Z\mathbf{S}$$

O transformare compusă se poate deci defini printr-o matrice de transformare **M** care este un produs (compunere) de matrice de transformări geometrice elementare. În exemplul dat, se calculează matricea **M** astfel:

$$\mathbf{M} = \mathbf{T}\mathbf{R}_Z\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}=\begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1{,}732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unul din vârfurile cubului, vârful P(1,1,1,1) se transformă în P'(8.732, 3.732,1,1):

$$\mathbf{P'}=\mathbf{M}\,\mathbf{P}=\begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1.732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}=\begin{bmatrix} 8.732 \\ 3.732 \\ 2 \\ 1 \end{bmatrix}$$

Ordinea de compunere a matricelor de transformare este definitorie pentru rezultatul transformării, dat fiind că produsul matricelor nu este comutativ.

Se poate verifica pe exemplul dat, prin inversarea ordinii transformării de rotație cu translația (fig. 3.4).

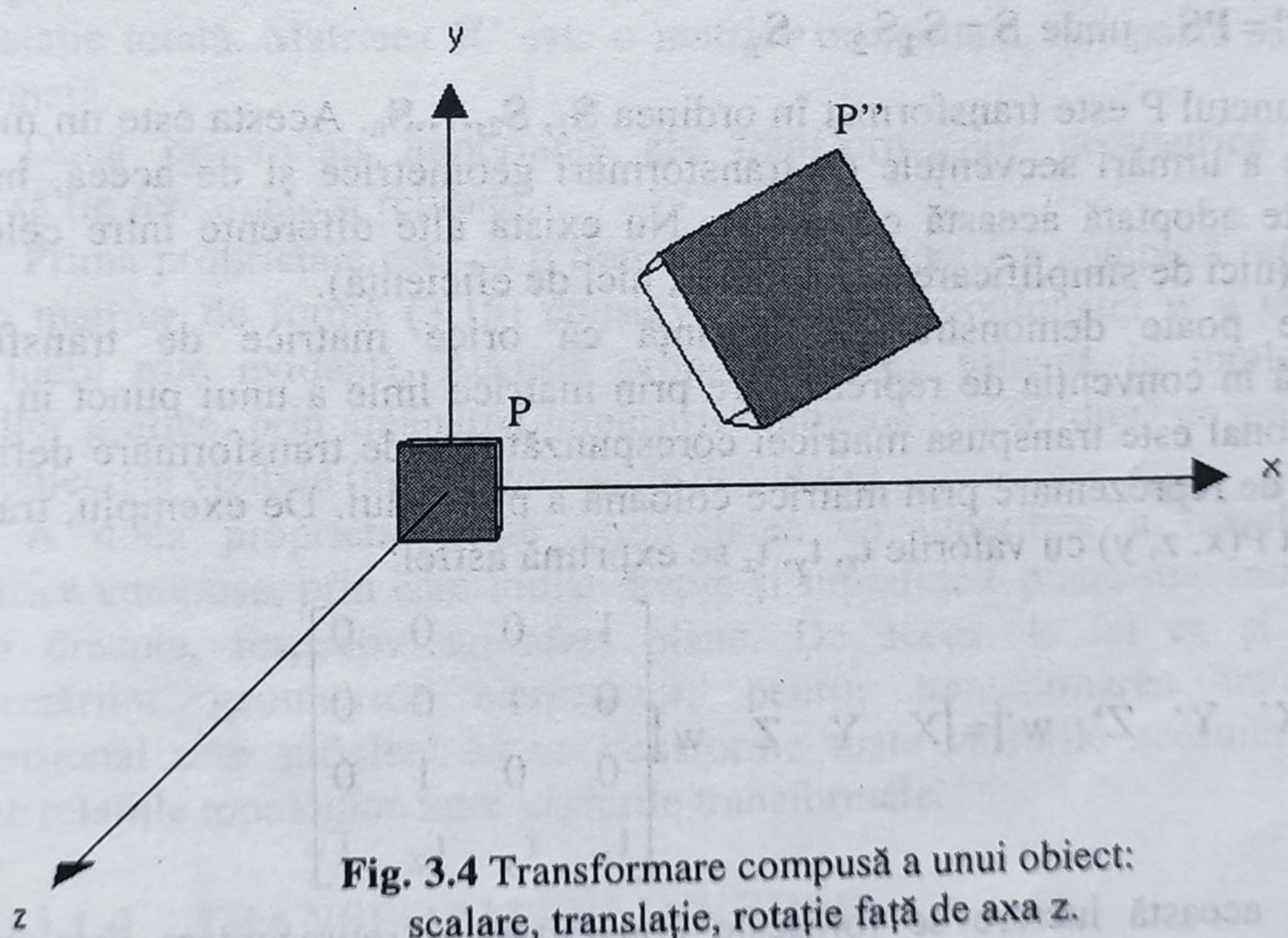


Fig. 3.4 Transformare compusă a unui obiect: scalare, translație, rotație față de axa z.

Se obține o matrice de transformare **M''** diferită de **M** și punctul transformat corespunzător P'', diferit de P':

$$\mathbf{M''}=\mathbf{R}_z\mathbf{T}\mathbf{S}=\begin{bmatrix} 1.732 & -1 & 0 & 6.928 \\ 1 & 1.732 & 0 & 4.330 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\quad \mathbf{P''}=\mathbf{M''}\mathbf{P}=\begin{bmatrix} 7.560 \\ 7.06 \\ 2 \\ 1 \end{bmatrix}$$

Convenţia de reprezentare a punctelor în spaţiu prin matrice coloană impune ordinea de înmulţire numită postmultiplicare (sau multiplicare la dreapta) a matricelor de transformare: se înmulţeşte matricea de transformare curentă cu matricea transformării următoare. Convenţia de terminologie (postmultiplicare sau multiplicare la dreapta) semnifică faptul că o nouă transformare se concatenează ca factor dreapta al produsului de matrice. Este important de reţinut faptul că ordinea de aplicare a transformărilor este de la dreapta la stânga din succesiunea de matrice ale unei compuneri. Mai precis, dacă un punct se transformă prin aplicarea succesivă a transformărilor definite prin matricele $\mathbf{M}_1$, $\mathbf{M}_2$,...., $\mathbf{M}_n$, matricea compusă de transformare este:

$$\mathbf{M} = \mathbf{M}_n \cdots \mathbf{M}_2 \mathbf{M}_1 \tag{3.12}$$

Se poate verifica că, dacă se adoptă convenţia de reprezentare a unui punct în spaţiul tridimensional printr-o matrice linie, atunci transformarea unui punct se obţine prin înmulţirea vectorului de poziţie al punctului cu matricea de transformare (premultiplicare sau multiplicare la stânga). În această situaţie, ordinea în care se aplică matricele compunente ale unei transformări compuse este de la stânga la dreapta:

$$\mathbf{P'} = \mathbf{PS}, \text{ unde } \mathbf{S} = \mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n \tag{3.13}$$

Punctul P este transformat în ordinea $\mathbf{S}_1, \mathbf{S}_2, \ldots \mathbf{S}_n$. Acesta este un mod mai natural de a urmări secvenţele de transformări geometrice şi de aceea, în unele lucrări este adoptată această convenţie. Nu există alte diferenţe între cele două convenţii (nici de simplificare a calculelor, nici de eficienţă).

Se poate demonstra cu uşurinţă că orice matrice de transformare elementară în convenţia de reprezentare prin matrice linie a unui punct în spaţiul tridimensional este transpusa matricei corespunzătoare de transformare definite în convenţia de reprezentare prin matrice coloană a punctului. De exemplu, translaţia unui punct P(x, z, y) cu valorile t_x, t_y, t_z se exprimă astfel:

$$[X' \quad Y' \quad Z' \quad w'] = [X \quad Y \quad Z \quad w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

În această lucrare se foloseşte convenţia de reprezentare prin matrice coloană a punctelor din spaţiu, dat fiind că aceasta este convenţia din biblioteca grafică OpenGL, care este folosită pentru exemplificarea operaţiilor grafice prezentate.

Un alt exemplu de transformare compusă este transformarea de rotaţie completă specificată prin trei rotaţii faţă de axele sistemului de coordonate. Cea mai obişnuită convenţie pentru ordinea de specificare a rotaţiilor este: ruliu cu unghiul ρ (după axa z), tangaj cu unghiul θ (după axa x) şi giraţie cu unghiul χ (faţă de axa y). În această situaţie matricea de rotaţie totală **R** are expresia:

$$\mathbf{R} = \mathbf{R}_Y(\chi)\mathbf{R}_X(\theta)\mathbf{R}_Z(\rho) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

Matricea **R** rezultată prin compunerea (înmulţirea) mai multor matrice de rotaţie este de asemenea ortogonală şi ortonormată.

O transformare complexă a unui obiect prin combinarea mai multor transformări elementare (scalări, rotaţii, translaţii) se poate exprima printr-o matrice de transformare **M** care are forma generală:

$$\mathbf{M} = \begin{bmatrix} r'_{11} & r'_{12} & r'_{13} & t_x \\ r'_{21} & r'_{22} & r'_{23} & t_y \\ r'_{31} & r'_{32} & r'_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

Submatricea stânga-sus **R'** de dimensiune 3 × 3 exprimă transformarea de rotaţie şi scalare totală, iar submatricea coloană dreapta **T** exprimă transformarea de translaţie totală. Matricea **R'** este o matrice ortogonală, dar poate să nu fie şi ortonormată.

Două proprietăţi importante ale transformărilor geometrice compuse trebuie să fie remarcate şi reţinute.

Prima proprietate este că o transformare geometrică compusă reprezentată printr-o matrice de forma (3.15) conservă valoarea coordonatei w a unui punct (acest lucru este evident). Această proprietate este folosită în implementarea operaţiilor grafice, prin amânarea împărţirii omogene (cu w) după ce s-au selectat numai obiectele vizibile (după operaţia de decupare).

A doua proprietate este proprietatea de liniaritate a transformărilor geometrice compuse, prin care liniile drepte şi suprafeţele plane sunt transformate în linii dreapte, respectiv suprafeţe plane. De aceea, la fel ca şi în cazul transformărilor geometrice elementare, pentru transformarea unui obiect tridimensional este suficient să se transforme toate vârfurile acestuia şi să se păstreze relaţiile topologice între vârfurile transformate.

3.1.4 TRANSFORMĂRI INVERSE

Fiind dată o transformare a unui punct P într-un punct P' definită printr-o matrice de transformare **M**, transformarea inversă, de la punctul P' la punctul P se obţine prin înmulţirea cu matricea inversă, $\mathbf{M}^{-1}$:

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{I},\ \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

unde **I** este matricea identitate.

Dat fiind că, în general, matricea de transformare **M** se obține printr-un produs de matrice de transformări elementare, matricea inversă $\mathbf{M}^{-1}$ se calculează prin produsul în ordine inversă a inverselor matricelor elementare componente:

$$\begin{aligned} \mathbf{M} &= \mathbf{M}_n \cdots \mathbf{M}_2 \mathbf{M}_1 \\ \mathbf{M}^{-1} &= \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_n^{-1} \\ \mathbf{M}\mathbf{M}^{-1} &= \mathbf{M}_n \cdots \mathbf{M}_2 \mathbf{M}_1 \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_n^{-1} = \mathbf{I} \end{aligned} \tag{3.17}$$

Relația (3.17) se demonstrează imediat, prin gruparea factorilor începând cu $\mathbf{M}_1 \mathbf{M}_1^{-1} = \mathbf{I}$.

Toate matricele de transformări elementare sunt matrice inversabile şi au următoarele expresii:

$$[\mathbf{T}(t_x, t_y, t_z)]^{-1} = \mathbf{T}(-t_x, -t_y, -t_z) = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$[\mathbf{S}(s_x, s_y, s_z)]^{-1} = \mathbf{S}(1/s_x, 1/s_y, 1/s_z) = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$[\mathbf{R}_X(\theta)]^{-1} = \mathbf{R}_X(-\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

$$[\mathbf{R}_Y(\chi)]^{-1} = \mathbf{R}_Y(-\chi) = \begin{bmatrix} \cos\chi & 0 & -\sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

$$[\mathbf{R}_Z(\rho)]^{-1} = \mathbf{R}_Z(-\rho) = \begin{bmatrix} \cos\rho & \sin\rho & 0 & 0 \\ -\sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

$$\mathbf{R}^{-1} = [\mathbf{R}_Y(\chi)\mathbf{R}_X(\theta)\mathbf{R}_Z(\rho)]^{-1} = \mathbf{R}_Z(-\rho)\mathbf{R}_X(-\theta)\mathbf{R}_Y(-\chi) \tag{3.23}$$

Datorită faptului că matricea de rotație este ortogonală şi ortonormată, inversa unei matrice de rotație este egală cu transpusa acesteia.

3.1.5 TRANSFORMAREA SISTEMELOR DE REFERINȚĂ

Interpretarea relaţiilor de transformări geometrice prezentate până acum a fost aceea de manevrare şi modificare a obiectelor într-un sistem de referinţă dat: obiectele sunt reprezentate într-un anumit sistem de referinţă prin coordonatele unei mulţimi de puncte ale acestora (vârfuri) şi ele pot fi deplasate, reorientate sau redimensionate prin aplicarea transformărilor geometrice corespunzătoare.

O altă interpretare care se poate da operaţiilor de transformări geometrice este aceea de schimbare a sistemului de referinţă.

Se consideră sistemul de referinţă Oxyz şi un nou sistem de referinţă O'x'y'z', a cărui origine O' este determinată în sistemul Oxyz prin coordonatele x_0,y_0,z_0 ale centrului O'. Sistemul de referinţă Oxyz este definit de versorii (vectori unitate) **i, j, k,** iar sistemul de referinţă O'x'y'z' de versorii **i', j', k'**.

Axa O'x' are cosinuşii directori c_{11}, c_{12}, c_{13} faţă de sistemul de referinţă Oxyz; axa O'y' are cosinuşii directori c_{21}, c_{22}, c_{23} faţă de sistemul de referinţă Oxyz; axa O'z' are cosinuşii directori c_{31}, c_{32}, c_{33} faţă de sistemul de referinţă Oxyz.

Matricea de transformare care descrie poziționarea sistemului O'x'y'z' relativ la sistemul de referinţă Oxyz este:

$$\mathbf{M}=\begin{bmatrix} c_{11} & c_{21} & c_{31} & x_0 \\ c_{12} & r_{22} & c_{32} & y_0 \\ c_{13} & c_{23} & c_{33} & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.24)$$

Fie un punct P definit prin coordonatele sale x,y,z în sistemul de referinţă Oxyz. Se demonstrează [Drag57], că în sistemul de referinţă O'x'y'z' acest punct (notat P') are coordonatele x',y',z', care se obţin prin înmulţirea matricei $\mathbf{M}^{-1}$ (care este inversa matricei **M** care defineşte poziționarea sistemului O'x'y'z' în sistemul Oxyz) cu matricea **P** de reprezentare a punctului P în sistemul de coordonate iniţial:

$$\mathbf{P'}=\mathbf{M}^{-1}\mathbf{P} \qquad (3.25)$$

Transformarea inversă, a unui punct **P'**(x',y',z') din sistemul de referinţă O'x'y'z' în punctul **P**(x,y,z) din sistemul de referinţă Oxyz se obţine prin înmulţire cu matricea de transformare **M**:

$$\mathbf{P}=\mathbf{M}\mathbf{P'} \qquad (3.26)$$

Se poate urmări cu mai multă uşurinţă această modalitate de transformare într-un caz simplu. Se consideră un sistem de referinţă Oxyz şi un alt sistem de referinţă O'x'y'z' care are originea **O'**(x_0,y_0,z_0) şi aceeaşi orientare a axelor de coordonate ca şi sistemul Oxyz.

Poziționarea sistemului de referinţă O'x'y'z' relativ la sistemul Oxyz este definită de matricea de transformare **M**; poziționarea sistemului Oxyz relativ la sistemul O'x'y'z' este definită de matricea de transformare inversă $\mathbf{M}^{-1}$:

$$\mathbf{M}=\begin{bmatrix}1 & 0 & 0 & x_0\\ 0 & 1 & 0 & y_0\\ 0 & 0 & 1 & z_0\\ 0 & 0 & 0 & 1\end{bmatrix};\ \mathbf{M}^{-1}=\begin{bmatrix}1 & 0 & 0 & -x_0\\ 0 & 1 & 0 & -y_0\\ 0 & 0 & 1 & -z_0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Submatricea de rotație componentă a matricei **M** este matricea unitate, dat fiind că sistemul de referință O'x'y'z' are axele cu aceeaşi orientare ca şi sistemul de referință. Într-adevăr, cosinuşii directori ai axelor O'x',O'y',O'z' față de axele sistemului de referință Oxyz sunt (1,0,0), (0,1,0), (0,0,1) şi matricea de rotație este matricea unitate.

Un punct oarecare **P**(x,y,z) în sistemul de referință Oxyz, se transformă în punctul **P'**(x',y',z') în sistemul de referință O'x'y'z' printr-o translație cu $(-x_0,-y_0,-z_0)$ deci :

$$\mathbf{P'}=\mathbf{M}^{-1}\mathbf{P};\quad \begin{bmatrix}x'\\ y'\\ z'\\ 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & -x_0\\ 0 & 1 & 0 & -y_0\\ 0 & 0 & 1 & -z_0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

Particularizarea pentru O'(4,2,0) şi P(6,3,0) este reprezentată în fig. 3.5. Punctul P'(2,1,0) este transformatul punctului P în sistemul de referință O'x'y'z'.

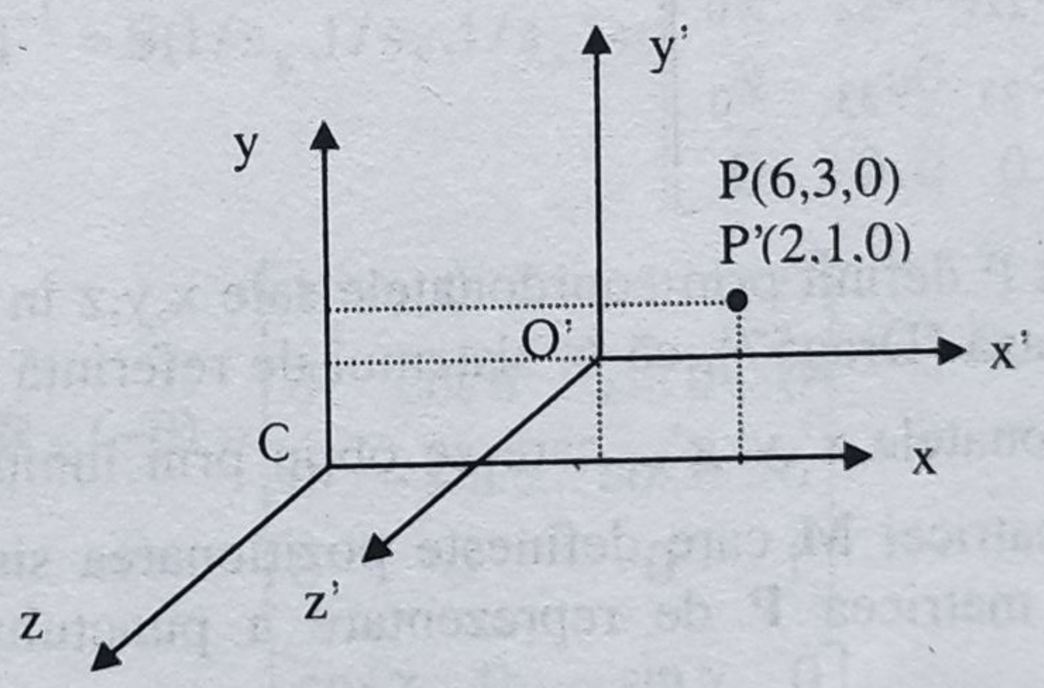


Fig.3.5 Transformarea sistemelor de referință.

În concluzie, aplicarea unei transformări definite printr-o matrice **M**, asupra unei mulțimi de puncte definite într-un sistem de referință Oxyz poate fi interpretată în mai multe moduri:

(a) Fiecare punct este modificat şi capătă o nouă poziționare în sistemul de referință Oxyz, conform cu matricea de transformare **M**.

(b) Fiecare punct este transformat din sistemul de referință inițial Oxyz într-un nou sistem de referință, O'x'y'z', a cărui poziție şi orientare relativ la sistemul de referință Oxyz este descrisă de matricea $\mathbf{M}^{-1}$.

(c) Fiecare punct este transformat din sistemul de referință inițial Oxyz într-un nou sistem de referință, O'x'y'z'. Poziția şi orientarea sistemului Oxyz relativ la sistemul de referință O'x'y'z' este descrisă de matricea **M**.

Şi încă o precizare: un sistem de referinţă nu este "materializat" în nici un fel într-un sistem grafic (în software sau hardware), ci este o convenţie cunoscută de programator sau utilizator. Trecerea de la un sistem de referinţă la altul se efectuează prin transformări aplicate punctelor (obiectelor): modificarea coordonatelor acestora le transferă dintr-un sistem de referinţă în altul. În acest sens, orice transformare geometrică poate fi considerată ca o schimbare a sistemului de referinţă. Dar, după cum se va observa în continuare, transformările geometrice sunt uneori considerate ca modificări ale obiectelor, iar în alte situaţii, ca schimbare a sistemului de referinţă. Este normal ca un începător în practica programării sistemelor grafice să pună o justificată întrebare: cum se poate şti care este interpretarea corectă a unei transformări? Răspunsul este că ambele interpretări sunt corecte, dar se alege aceea care permite urmărirea cea mai directă a unui raţionament pentru calculul unor transformări complexe. Nu există "reţete" unice şi sigure care să poată fi aplicate fără greşeală în orice situaţie, deci experienţa de proiectare îşi spune întotdeauna cuvântul.

În dezvoltarea sistemelor de vizualizare se folosesc, totuşi, câteva sisteme de referinţă bine definite, care permit specificarea cea mai simplă şi eficientă a operaţiilor de redare a obiectelor (scenelor). Unul dintre acestea, sistemul de referinţă universal, a fost deja introdus. Alte sisteme de referinţă folosite în grafica tridimensională vor fi definite pe parcursul lucrării. Sistemele de referinţă intermediare, care apar în descrierea unor transformări complexe, sunt interpretări ale operaţiilor de transformări geometrice care permit urmărirea unui raţionament de calcul. Exemplul următor evidenţiază acest aspect.

■ **Exemplul 3.1**

Se consideră sistemul de coordonate Oxyz şi o matrice de transformare $\mathbf{R}_X(\pi/2)$ care realizează o rotaţie cu unghiul $\pi/2$ în raport cu axa x:

$$\mathbf{R}_X(\pi/2)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Această transformare poate fi interpretată ca o transformare prin rotaţie cu 90° în raport cu axa x a fiecărui punct P(x,y,z), în punctul transformat P'(x',y',z'). În fig.3.6 (a) este considerat punctul P(0,1,1), care se transformă în punctul P'(0,–1,1). În fig. 3.6(b) este reprezentată transformarea echivalentă a sistemului de coordonate Oxyz în sistemul de coordonate O'x'y'z' folosind matricea de transformare inversă corespunzătoare, $\mathbf{R}_X(-\pi/2)$: axa y se transformă în axa z', iar axa z se transformă în axa –y'. Punctul P, de coordonate (0,1,1) în sistemul Oxyz, se transformă în punctul P' de coordonate (0,–1,1) în sistemul O'x'y'z'.

Se pot verifica uşor rotaţiile cu un unghi $\pi/2$ în raport cu celelalte axe de coordonate. Rotaţia cu $\pi/2$ faţă de axa y transformă sistemul de coordonate într-un sistem în care axa z se schimbă în axa x, iar axa x se schimbă în axa –z. Rotaţia cu

$\pi/2$ față de axa z transformă sistemul de coordonate într-un sistem în care axa x se schimbă în axa y, iar axa y se transformă în axa –x.

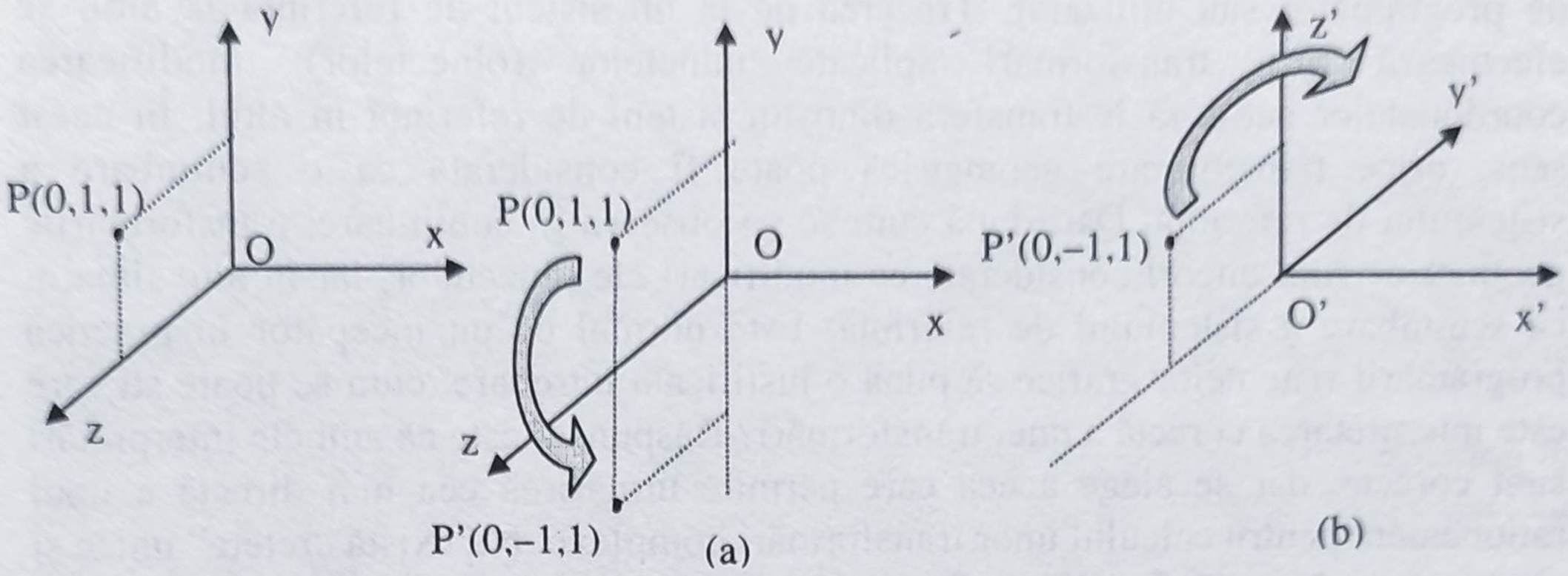


Fig. 3.6 Rotație cu $\pi/2$ în raport cu axa x:
(a) rotația punctelor cu $\pi/2$; (b) rotația sistemului de coordonate cu $-\pi/2$.

Pentru deducerea matricelor unor transformări complexe se procedează prin reducerea acestora la cazuri mai simple, pentru care se cunosc expresiile de calcul, folosind transformări primitive succesive. Inițial se aplică o transformare ajutătoare pentru aducerea obiectelor într-o poziție sau sistem de referință adecvat, se execută transformarea dorită, după care se revine la poziția sau sistemul de referință inițial prin transformarea inversă celei aplicate inițial. Câteva exemple prezentate în continuare vor preciza acest mod de realizare a transformărilor geometrice compuse.

3.1.5.1 Rotația față de o axă paralelă cu una din axele sistemului de referință

Un exemplu ilustrativ de compunere a transformărilor geometrice este calculul rotației în raport cu o axă paralelă cu una din axele sistemului de referință.

Se consideră o dreaptă D paralelă cu axa z a sistemului, care intersectează planul Oxy în punctul $I(t_x,t_y,0)$. Transformarea de rotație a obiectelor față de această axă nu poate fi realizată folosind matricea dedusă în paragraful precedent (relația 3.10), care presupune rotația față de o axă a sistemului de coordonate.

De aceea, se aplică mai întâi tuturor punctelor o transformare ajutătoare, translația cu $\mathbf{T}(-t_x,-t_y,0)$. Acestă transformare poate fi intrepretată în două moduri: ca o schimbare a sistemului de referință Oxyz în sistemul O'x'y'z' , în care punctul I este transformat în punctul I'(0,0,0), sau ca o modificare a poziției tuturor punctelor din spațiu prin care punctul I este adus în originea sistemului de referință Oxyz.

Dacă se consideră prima interpretare, operațiile se continuă în noul sistem de referință O'x'y'z', în care dreapta d se suprapune peste axa z', deci se poate obține rotația dorită cu un unghi ρ folosind matricea de rotație (3.10). După aceasta, se revine la sistemul de referință inițial Oxyz, printr-o transformare

inversă, $\mathbf{T}(t_x, t_y, 0)$. Rezultă matricea compusă de rotaţie cu unghiul ρ faţă de o dreaptă paralelă cu axa z care interectează planul Oxy în punctul $I(t_x,t_y,0)$:

$$\mathbf{R_D} = \mathbf{T}(t_x, t_y, 0)\mathbf{R_Z}(\rho)\mathbf{T}(-t_x, -t_y, 0)$$

$$\mathbf{R_D} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\rho & -\sin\rho & 0 & 0 \\ \sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_D} = \begin{bmatrix} \cos\rho & -\sin\rho & 0 & t_x(1-\cos\rho) + t_y \sin\rho \\ \sin\rho & \cos\rho & 0 & t_y(1-\cos\rho) - t_x \sin\rho \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.27)$$

Acest proces este descris în fig. 3.7 (a) pentru un unghi de 30 grade, cu reprezentarea unei proiecţii în planul Oxy.

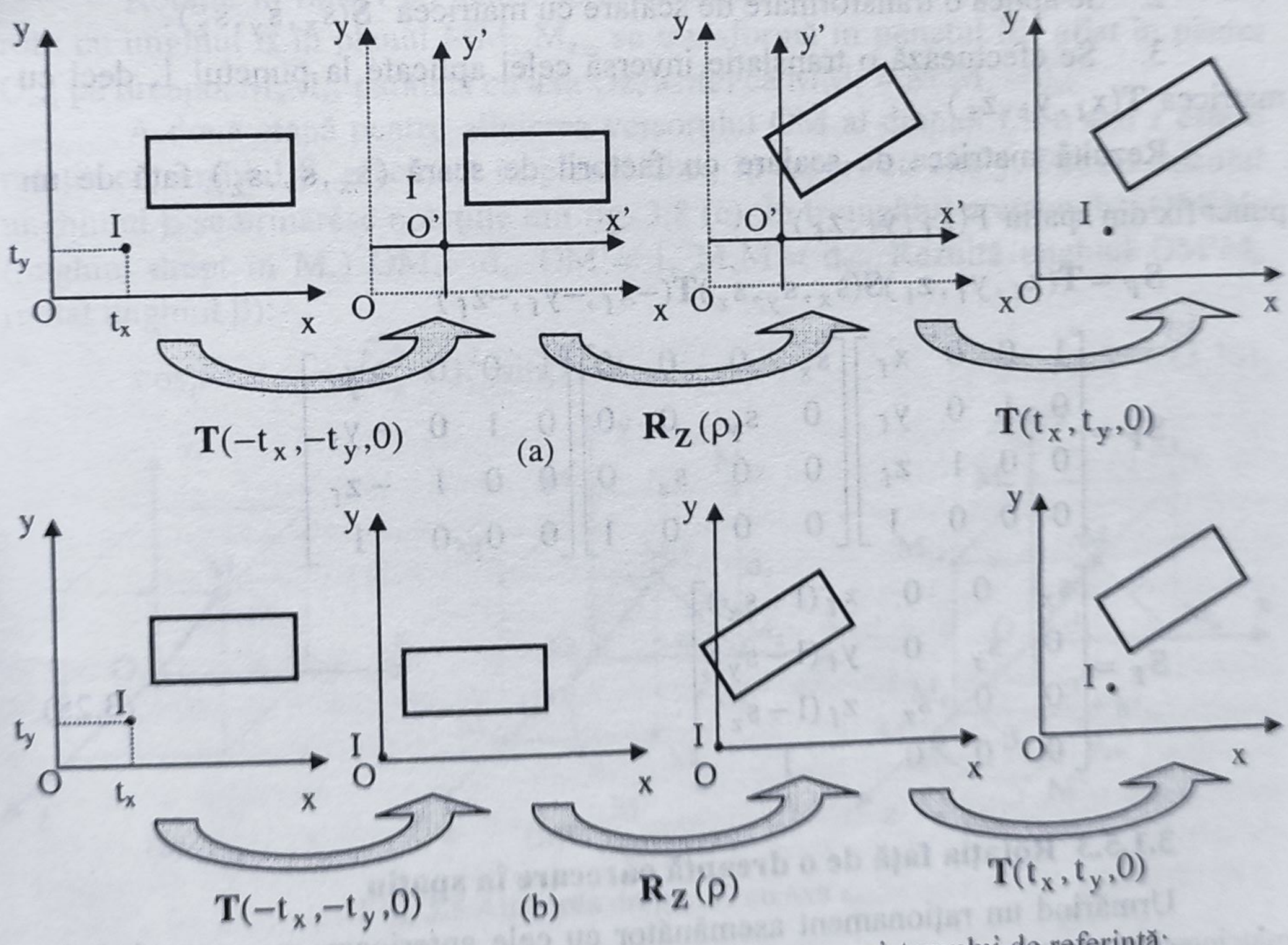


Fig. 3.7 Rotaţia faţă de o dreaptă paralelă ca axa z a sistemului de referinţă:
(a) considerând transformarea sistemelor de referinţă;
(b) considerând transformarea punctelor.

În fig. 3.7 (b) se prezintă aceleaşi transformări succesive necesare pentru realizarea rotaţiei în raport cu o dreaptă paralelă cu axa z a sistemului, considerând

că transformarea inițială $\mathbf{T}(-t_x,-t_y,0)$ transformă toate punctele din sistemul de referință Oxyz astfel încât punctul I se suprapune peste originea O. Se remarcă faptul că, în această situație, este doar o diferență de interpretare, operațiile și rezultatul acestora fiind identic.

3.1.5.2 Scalarea față de un punct oarecare în spațiu

Matricea de scalare din relația (3.4) efectuează scalarea față de originea sistemului de coordonate, prin care componentele x, y, z ale vectorului de poziție OP al punctului P sunt multiplicate fiecare cu factorul de scalare corespunzător. Se poate defini o scalare față de un punct fix $F(x_f, y_f, z_f)$ din spațiu, prin care componentele vectorului FP sunt multiplicate cu factorii de scalare corespunzători.

Matricea de scalare față de un punct oarecare se poate deduce printr-o metodă asemănătoare celei prezentate anterior, prin compunerea a trei transformări elementare deja definite.

1. Se execută o translație cu $\mathbf{T}(-x_f,-y_f,-z_f)$, prin care punctul fix F se suprapune peste originea sistemului de coordonate.
2. Se aplică o transformare de scalare cu matricea $\mathbf{S}(s_x,s_y,s_z)$.
3. Se efectuează o translație inversă celei aplicate la punctul 1, deci cu matricea $\mathbf{T}(x_f, y_f, z_f)$.

Rezultă matricea de scalare cu factorii de scară (s_x,s_y,s_z) față de un punct fix din spațiu $F(x_f, y_f, z_f)$:

$$\mathbf{S_F} = \mathbf{T}(x_f, y_f, z_f)\mathbf{S}(s_x,s_y,s_z)\mathbf{T}(-x_f,-y_f,-z_f)$$

$$\mathbf{S_F} = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S_F} = \begin{bmatrix} s_x & 0 & 0 & x_f(1-s_x) \\ 0 & s_y & 0 & y_f(1-s_y) \\ 0 & 0 & s_z & z_f(1-s_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

3.1.5.3 Rotația față de o dreaptă oarecare în spațiu

Urmărind un raționament asemănător cu cele anterioare, se poate deduce matricea de rotație cu un unghi δ față de o dreaptă oarecare D din spațiu, dată printr-un punct $D_0(x_0,y_0,z_0)$ și vectorul unitate (versorul) **d**, având cosinușii directori d_x, d_y, d_z ($\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$). Pașii de transformare sunt următorii:

1. Translația cu matricea $\mathbf{T}(-x_0,-y_0,-z_0)$, prin care punctul D_0 ajunge în originea sistemului de referință.

2. Alinierea dreptei D cu una din axele sistemului de referinţă, de exemplu, axa z. Alinierea cu una din axele de coordonate ale sistemului se efectuează prin două rotaţii în raport cu celelalte două axe.

Mai întâi se efectuează o rotaţe în raport cu axa x, cu un unghi α, astfel ca dreapta D să ajungă în planul Oxz. Pentru calculul unghiului α se consideră notaţiile din fig. 3.8 (b). OM este versorul dreptei D după translaţia efectuată în pasul precedent. M_x, M_y, M_z sunt intersecţiile planelor paralele cu planele sistemului de coordonate care trec prin punctul M, cu axele corespunzătoare (M_x este intersecţia planului paralel cu planul Oyz care trece prin M, cu axa x, etc.). M_{xy}, M_{yz}, M_{zx} sunt proiecţiile punctului M pe planele O_{xy}, O_{yz} şi, respectiv, O_{zx}. Se pot scrie următoarele relaţii:

$$OM_x = d_x ; OM_y = d_y ; OM_z = d_z$$

$$MM_x = d_{yz} = \sqrt{d_y^2 + d_z^2}$$

$$\cos\alpha = d_z / d_{yz} = d_z \Big/ \sqrt{d_y^2 + d_z^2} ; \sin\alpha = d_y / d_{yz} = d_y \Big/ \sqrt{d_y^2 + d_z^2} \qquad (3.29)$$

Rotaţia în raport cu axa x se efectuează în planul $MM_{xy}M_{zx}$. Punctul M, rotit cu unghiul α în planul $MM_{xy}M_{zx}$, se transformă în punctul M' aflat în planul O_{zx}, pe dreapta M_xM_{zx} paralelă cu axa Oz, astfel că $MM_x = M'M_x = d_{yz}$.

A doua etapă pentru alinierea versorului OM al dreptei D cu axa z este o rotaţie cu unghiul β, efectuată în planul Ozx, în raport cu axa y. Pentru calculul unghiului β se urmăresc notaţiile din fig. 3.8 (c). În triunghiul dreptunghic $OM'M_x$ (unghiul drept în M_x) $OM_x = d_x$, $OM' = 1$, $M_xM' = d_{yz}$. Rezultă unghiul $OM'M_x$ (notat unghiul β):

$$\cos\beta = d_{yz} = \sqrt{d_y^2 + d_z^2} ; \sin\beta = d_x \qquad (3.30)$$

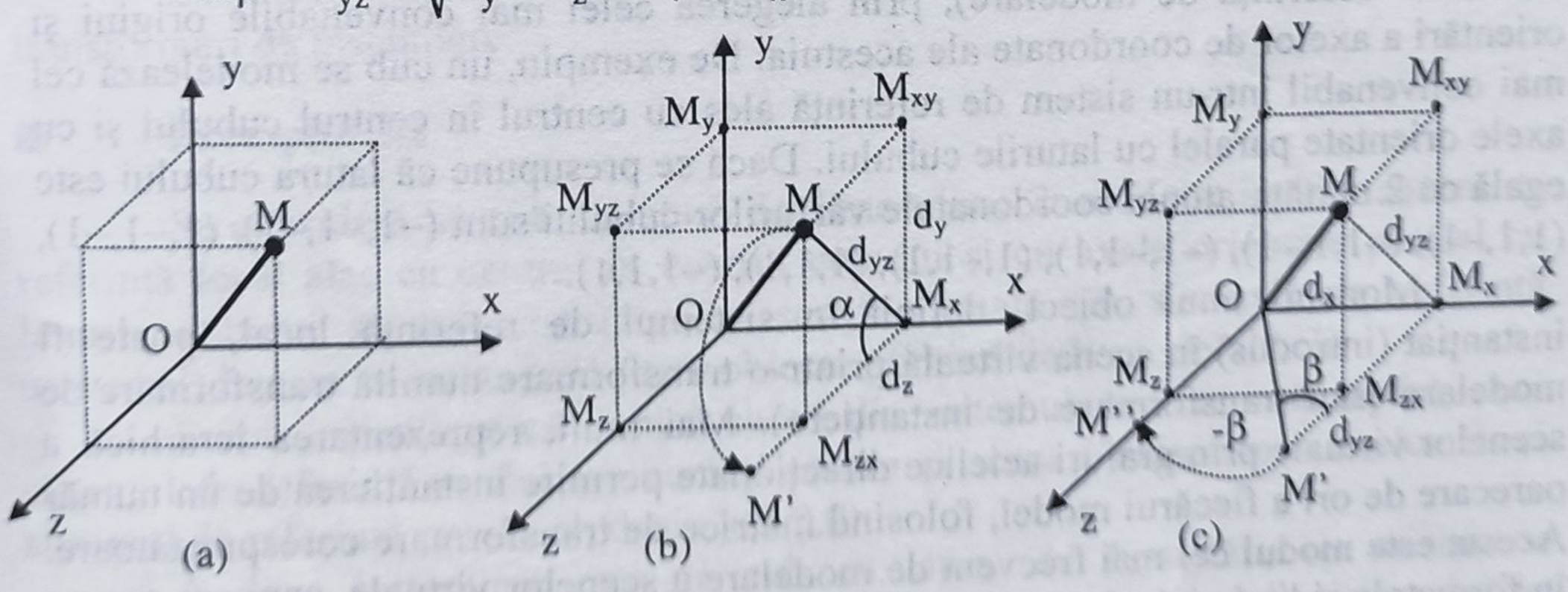


Fig. 3.8 Alinierea dreptei D cu axa z.

Rotaţia dreptei OM' în planul Ozx faţă de axa Oy transformă punctul M' în punctul M'' pe axa Oz printr-o rotaţie cu unghiul de valoare β în sens invers sensului pozitiv de rotaţie după axa y (a se revedea fig. 3.3) Această transformare de aliniere a dreptei D cu axa z se exprimă matriceal astfel:

$$\mathbf{A}_Z = \mathbf{R}_Y(-\beta)\mathbf{R}_X(\alpha)$$

3. În acest pas se execută rotaţia dorită, cu unghiul δ în raport cu dreapta D, care este aliniată cu axa z a sistemului, deci se poate scrie:

$$\mathbf{R_Z}(\delta)=\begin{bmatrix}1 & \cos\delta & -\sin\delta & 0\\ 0 & \sin\delta & \cos\delta & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

4. În pasul al patrulea se execută transformarea inversă celei de la pasul 2, deci cu o matrice $\mathbf{A_Z^{-1}}=\mathbf{R_X}(-\alpha)\mathbf{R_Y}(\beta)$.

5. În ultimul pas se execută transformarea de translaţie inversă faţă de cea executată în pasul 1, cu matricea $\mathbf{T}(x_0,y_0,z_0)$.

Rezultă matricea compusă de rotaţie faţă de dreapta D cu unghiul δ:

$$\mathbf{R_D}=\mathbf{T}(x_0,y_0,z_0)\mathbf{R_X}(-\alpha)\mathbf{R_Y}(\beta)\mathbf{R_Z}(\delta)\mathbf{R_Y}(-\beta)\mathbf{R_X}(\alpha)\mathbf{T}(-x_0,-y_0,-z_0) \quad (3.31)$$

Valorile unghiurilor α şi β rezultă din cosinuşii directori ai axei de rotaţie D şi au valorile date de relaţiile (3.29) şi (3.30).

3.1.5.4 Transformarea de modelare

O scenă virtuală este compusă dintr-un număr oarecare de obiecte tridimensionale amplasate în diferite poziţii în scenă. Tehnica de modelare a scenelor de dimensiuni mari (ca număr de obiecte sau ca volum ocupat de scenă) se bazează pe amplasarea modelelor obiectelor tridimensionale în sistemul de referinţă universal al scenei virtuale.

Fiecare obiect este modelat într-un sistem de referinţă local (numit şi sistem de referinţă de modelare), prin alegerea celei mai convenabile origini şi orientări a axelor de coordonate ale acestuia. De exemplu, un cub se modelează cel mai convenabil într-un sistem de referinţă ales cu centrul în centrul cubului şi cu axele orientate paralel cu laturile cubului. Dacă se presupune că latura cubului este egală cu 2 unităţi, atunci coordonatele vârfurilor cubului sunt (–1,–1,–1), (1,–1,–1), (1,1,–1), (–1,1,–1), (–1,–1,1), (1,–1,1), (1,1,1), (–1,1,1).

Modelul unui obiect, definit în sistemul de referinţă local, poate fi instanţiat (introdus) în scena virtuală printr-o transformare numită transformare de modelare (sau transformare de instanţiere). Mai mult, reprezentarea ierarhică a scenelor virtuale prin grafuri aciclice direcţionate permite instanţierea de un număr oarecare de ori a fiecărui model, folosind matrice de transformare corespunzătoare. Acesta este modul cel mai frecvent de modelare a scenelor virtuale, care are suport în formatele şi limbajele de descriere a scenelor virtuale, cum este limbajul VRML.

În modelarea ierarhică a scenelor virtuale, operaţia de bază este transformarea de modelare prin care toate punctele unui obiect (model tridimensional) sunt transformate din sistemul de referinţă local într-un alt sistem de referinţă, care poate fi sistemul de referinţă universal sau un alt sistem intermediar. Pentru început se consideră cea mai simplă transformare de modelare, dintr-un sistem de referinţă local în sistemul de referinţă universal.

În mod obişnuit, prima transformare care se aplică modelului este transformarea de scalare cu o matrice de scalare $\mathbf{S}(s_x,s_y,s_z)$, executată în sistemul de referinţă local (faţă de originea acestuia), prin care obiectul este adus la dimensiunile dorite ale instanţei respective.

Poziţia unei instanţe a obiectului în scenă se defineşte printr-o matrice de transformare conţinând submatricea de rotaţie **R**, care defineşte orientarea sistemului de referinţă local O'x'y'z' faţă de sistemul de referinţă universal Oxyz (al scenei virtuale) şi submatricea de translaţie **T**, care defineşte poziţia originii $O'(x_0,y_0,z_0)$ a sistemului de referinţă local în sistemul universal. Submatricea de rotaţie **R** poate fi specificată fie prin cosinuşii directori (c_{11}, c_{12}, c_{13}), (c_{21}, c_{22}, c_{23}), (c_{31}, c_{32}, c_{33}) ai axelor sistemului local faţă de sistemul de referinţă universal, fie printr-o succesiune de rotaţii faţă de axele de coordonate ale sistemului de referinţă local:

$$\mathbf{R}=\begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & r_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{sau} \quad \mathbf{R}=\mathbf{R_Y}(\chi)\mathbf{R_X}(\theta)\mathbf{R_Z}(\rho)$$

Rezultă matricea de transformare de modelare (instanţiere):

$$\mathbf{M_I}=\mathbf{T}(x_0,y_0,z_0)\mathbf{R}\,\mathbf{S}(s_x,s_{y,}s_z) \tag{3.32}$$

Un punct P al obiectului în sistemul de referinţă model este transformat în punctul P_I în sistemul de referinţă universal prin instanţierea:

$$\mathbf{P_I}=\mathbf{M_I}\mathbf{P} \tag{3.33}$$

Exemplul următor ilustrează crearea unei scene virtuale simple prin transformări de instanţiere.

■ **Exemplul 3.2**

Se consideră un cub ca latura de dimensiune 2, modelat într-un sistem de referinţă local ales cu centrul în centrul cubului şi cu axele orientate paralel cu laturile cubului şi se construieşte o scenă virtuală (în sistemul de referinţă universal) în care se instanţiază patru obiecte pornind de la modelul cub.

Prima instanţiere este chiar cubul din sistemul de referinţă model, deci sistemul de referinţă model are acelaşi centru şi aceeaşi orientare a axelor ca şi sistemul de referinţă universal; deci $C_1(0,0,0)$.

A doua instanţiere este un cub cu latura de 4 unităţi, cu orientarea sistemului de referinţă local cu axele de coordonate paralele cu axele sistemului de referinţă al scenei, amplasat cu centrul în punctul $C_2(0,-8,0)$.

A treia instanţiere este un paralelipiped dreptunghic, cu laturile de dimensiuni ,4,2,4, amplasat cu centrul în poziţia $C_3(8,0,0)$. Sistemul de referinţă local are axa z' paralelă cu axa z a sistemului universal, iar axele x' şi y' sunt rotite cu un unghi de $\pi/4$ faţă de axa x şi, respectiv, y a sistemului universal.

A patra instanţiere este un paralelipiped dreptunghic, cu laturile de dimensiuni 4,2,4, amplasat cu centrul în poziţia $C_4(-8,0,0)$. Sistemul de referinţă local are axa z' paralelă cu axa z a sistemului universal, iar axele x' şi y' sunt rotite cu un unghi de $-\pi/4$ faţă de axa x şi, respectiv, y a sistemului universal.

Pentru crearea scenei se calculează pe rând cele trei matrice de instanţiere.

Matricea de instanţiere $\mathbf{M}_1$ este chiar matricea identitate: $\mathbf{M}_1 = \mathbf{I}$.

Pentru calculul matricei de instanţiere $\mathbf{M}_2$, se deduc în mod foarte simplu matricele $\mathbf{S}_2=\mathbf{S}(2,2,2)$, $\mathbf{R}_2=\mathbf{I}$, şi $\mathbf{T}_2=\mathbf{T}(0,-8,0)$, deci: $\mathbf{M}_2 = \mathbf{T}(0,-8,0)\,\mathbf{S}(2,2,2)$.

Pentru calculul matricei de instanţiere $\mathbf{M}_3$, se poate scrie $\mathbf{S}_3=\mathbf{S}(2,1,2)$, $\mathbf{R}_3=\mathbf{R}_Z(\pi/4)$, $\mathbf{T}_3=\mathbf{T}(8,0,0)$, deci: $\mathbf{M}_3 = \mathbf{T}(8,0,0)\,\mathbf{R}_Z(\pi/4)\,\mathbf{S}(2,1,2)$.

Pentru calculul matricei de instanţiere $\mathbf{M}_4$, se poate scrie: $\mathbf{S}_4=\mathbf{S}(2,1,2)$, $\mathbf{R}_4=\mathbf{R}_Z(-\pi/4)$, $\mathbf{T}_4=\mathbf{T}(-8,0,0)$, $\mathbf{M}_4 = \mathbf{T}(-8,0,0)\,\mathbf{R}_Z(-\pi/4)\,\mathbf{S}(2,1,2)$.

Scena virtuală rezultată este redată în fig. 3.9. Axele de coordonate reprezentate în imagine sunt axele sistemului de referinţă universal. În centrul sistemului de referinţă universal este reprezentat primul cub, cu aceleaşi dimensiuni ca cele ale modelului.

Cubul din partea de jos a imaginii se obţine prin transformarea de instanţiere $\mathbf{P}_2 = \mathbf{M}_2\,\mathbf{P}$; paralelipipedul din dreapta se obţine prin transformarea de instanţiere $\mathbf{P}_3 = \mathbf{M}_3\,\mathbf{P}$; paralelipipedul din stânga se obţine prin transformarea de instanţiere $\mathbf{P}_4 = \mathbf{M}_4\,\mathbf{P}$.

Reprezentarea de mai jos conţine şi o transformare de proiecţie perspectivă, pentru percepţia adâncimii care va fi explicată ulterior. Una din feţele cubului (cea mai apropiată de observator) este desenată ca suprafaţă de o culoare gri; celelalte feţe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare "cadru de sârmă" – *wireframe*)

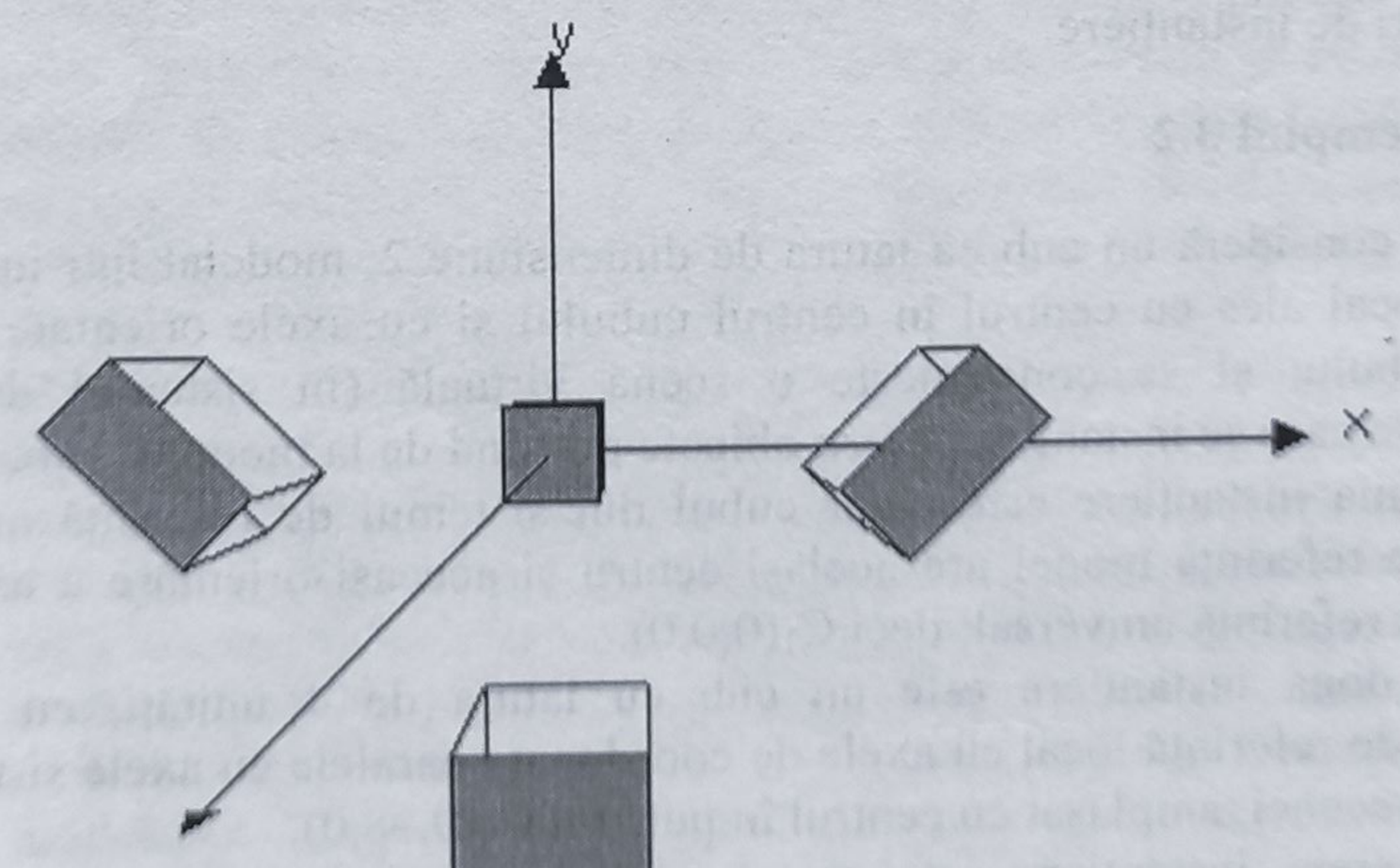


Fig. 3.9 Scenă virtuală rezultată prin instanţierea mai multor obiecte.

3.1.6 ALTE TRANSFORMĂRI GEOMETRICE ÎN SPAȚIU

Printre alte transformări în spaţiul tridimensional care se pot aplica obiectelor, cele mai importante sunt transformarea de simetrie şi transformarea de forfecare.

Transformarea de simetrie. O transformare de simetrie (oglindire) a unui punct P faţă de un plan π calculează un punct P' aflat pe o dreaptă Δ perpendiculară pe planul π, la o distanţă egală cu distanţa lui P faţă de planul dat. Pentru cazul particular în care planul de simetrie este unul din planele sistemului de coordonate, simetricul unui punct este punctul care are coordonatele pe cele două axe ale planului egale şi coordonata pe cealaltă axă a sistemului de coordonate cu valoare negată. Se pot deduce simplu matricele de transformare de simetrie relative la planele sistemului de referinţă:

$$\mathbf{S_{XY}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{S_{YZ}} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{S_{ZX}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.34)$$

Transformarea de simetrie faţă de un plan oarecare se poate calcula folosind o secvenţă de transformări elementare.

Dacă se consideră transformarea de simetrie ca o transformare aplicată unui sistem de coordonate dat, sistemul de coordonate rezultat are convenţia de orientare a axelor inversă faţă de convenţia iniţială. De exemplu, sistemul drept Oxyz se transformă prin simetria $\mathbf{S_{xy}}$ într-un sistem de coordonate orientat după regula mâinii stângi (fig. 3.10).

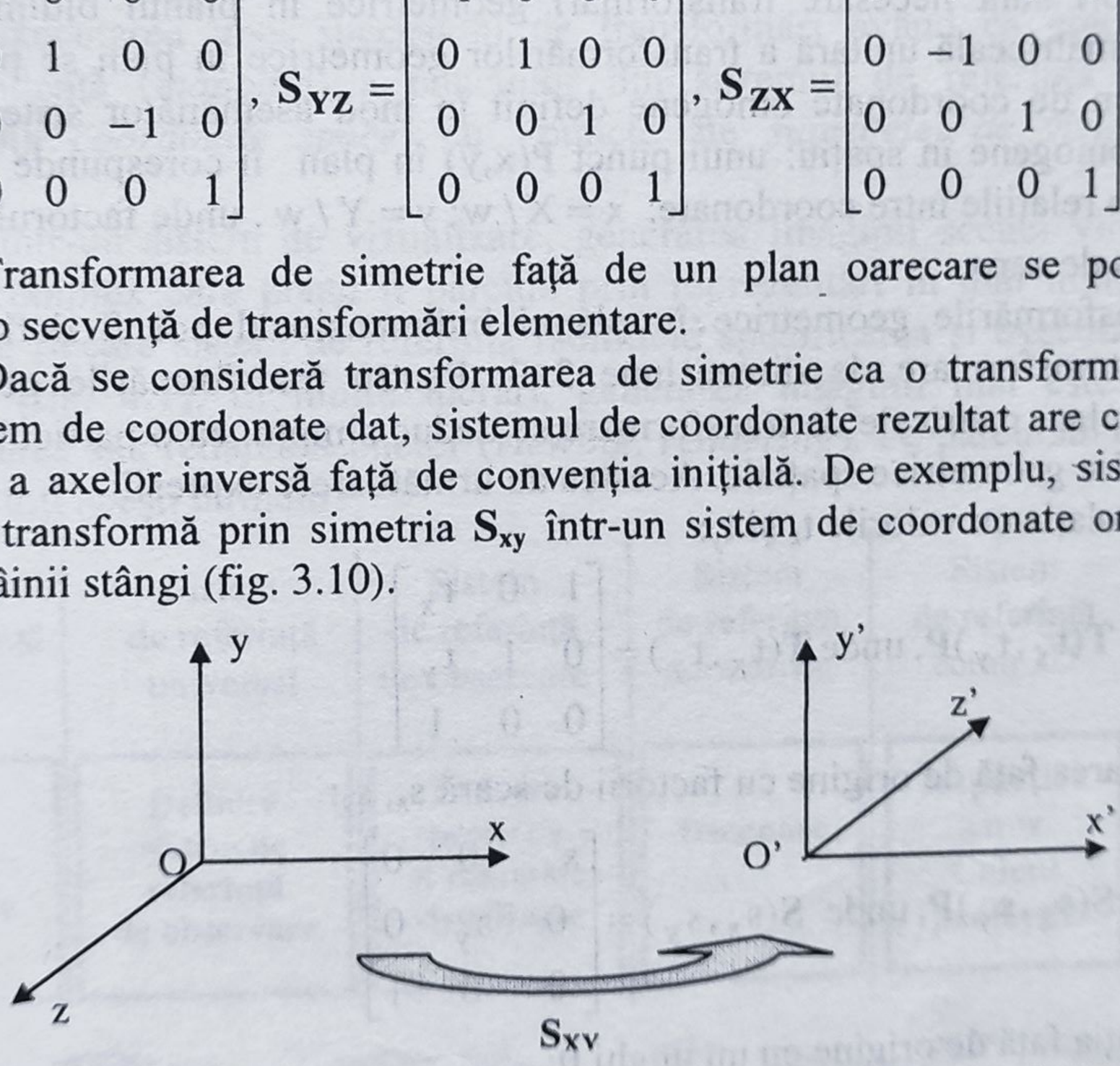


Fig. 3.10 Transformarea de simetrie $\mathbf{S_{XY}}$ transformă sistemul de coordonate drept Oxyz în sistemul de coordonate stâng O'x'y'z'.

Transformarea de forfecare (*shear*) este o transformare care modifică forma şi dimensiunea obiectului transformat. Transformările de forfecare cele mai simple sunt cele definite faţă axele sistemului de coordonate.

Forfecarea după axa x a sistemului de coordonate modifică un punct P(x,y,z) în punctul P'(x',y',z'), astfel că:

$$x' = x + ay + bz;\ y' = y;\ z' = z$$

În forma matriceală în coordonate omogene, forfecarea după axa x se scrie astfel:

$$\mathbf{P'} = \mathbf{F_X P}\text{, unde } \mathbf{F_X}(a,b) = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

În mod similar se definesc matricele de forfecare pe axele y şi z. Transformarea de forfecare este folosită în definirea sistemelor de proiecție.

3.2 TRANSFORMĂRI GEOMETRICE ÎN PLAN

Uneori sunt necesare transformări geometrice în planul bidimensional. Exprimarea matriceală unitară a transformărilor geometrice în plan se poate face într-un sistem de coordonate omogene definit în mod asemănător sistemului de coordonate omogene în spațiu: unui punct P(x,y) în plan îi corespunde un punct P(X,Y,w), cu relațiile între coordonate: $x = X/w;\ y = Y/w$, unde factorul de scară w este diferit de zero.

Transformările geometrice în planul bidimensional pot fi definite prin matrice de transformare de dimensiune 3×3. Pentru transformările geometrice primitive în plan, matricele de transformare se deduc simplu, ca o particularizare a transformărilor geometrice spațiale. Acestea au următoarele expresii:

Translația cu valorile t_x şi t_y:

$$\mathbf{P'} = \mathbf{T}(t_x, t_y)\mathbf{P}\text{, unde } \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.36}$$

Scalarea față de origine cu factorii de scară s_x, s_y:

$$\mathbf{P'} = \mathbf{S}(s_x, s_y)\mathbf{P}\text{, unde } \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

Rotația față de origine cu un unghi θ:

$$\mathbf{P'} = \mathbf{R}(\theta)\mathbf{P}\text{, unde } \mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

Transformări complexe în plan se pot defini, la fel ca şi în spațiu, prin compunerea (produs de matrice) de transformări primitive. Toate transformările între sisteme de referință în plan pot fi considerate cazuri particulare ale transformărilor corespunzătoare în spațiu.

SISTEME DE VIZUALIZARE

Un sistem de vizualizare este definit printr-o colecție de sisteme de referință, convenții de reprezentare şi relații (matriceale) de transformare care permit executarea unei succesiuni de transformări având ca efect punerea în corespondență (*mapping*) a punctelor din sistemul de referință de modelare (*modelling coordinate space*) cu puncte ale *suprafeței de vizualizare* (*view surface*).

Într-un sistem de vizualizare, generarea imaginii scenei virtuale este un proces complex care poate fi parcurs prin reprezentări în mai multe sisteme de referință, fiecare sistem de referință facilitând specificarea şi executarea anumitor operații (fig. 4.1). În multe lucrări, generarea imaginii mai este denumită şi vizualizarea sau redarea scenelor (*viewing*, *rendering*). Pe parcursul lucrării vor fi utilizați toți aceşti termeni.

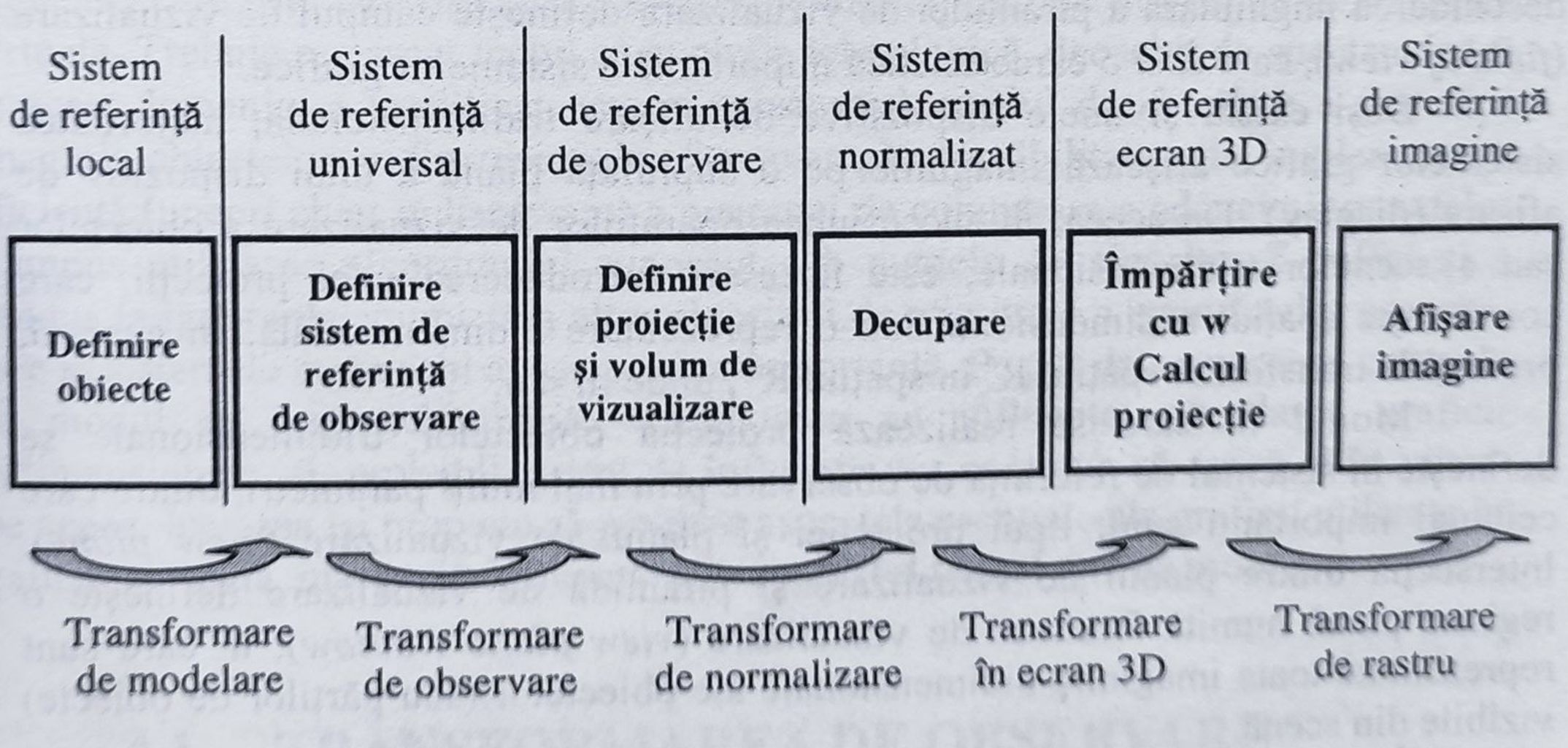


Fig. 4.1 Succesiunea operațiilor de vizualizare.

Obiectele tridimensionale componente ale scenei sunt modelate şi reprezentate în sisteme de referință locale (sau de modelare), care au originea şi orientarea alese cel mai convenabil pentru reprezentarea obiectelor respective.

Pentru construirea scenei, obiectele specificate în sistemul de referință de modelare sunt instanțiate prin aplicarea unei succesiuni de transformări geometrice, care constitue transformarea de modelare (descrisă în § 3.1.5.4). Prin transformarea de modelare se obține descrierea obiectelor într-un sistem de referință unic al scenei virtuale, numit sistemul de referință universal. În sistemul de referință universal mai sunt definite şi alte elemente necesare redării realiste a scenelor, cum sunt sursele de lumină sau traiectoriile obiectelor mobile în scenă.

În *sistemul de referință universal* se defineşte un sistem de referință de observare (*view reference*), care specifică punctul şi direcția din care este privită (observată) scena. Este evident că, pentru aceeaşi scenă, imaginea care se redă pe display depinde de punctul şi direcția din care este observată scena, deci de felul în care este definit sistemul de referință de observare (care mai este denumit şi sistem de referință ochi – *eye coordinate space*). Transformarea obiectelor din sistemul de referință universal în sistemul de referință de observare este numită transformare de observare (sau de vizualizare – *view transformation*; s-a evitat, însă, această traducere, datorită suprautilizării termenului de vizualizare).

În *sistemul de referință de observare* se poate face o analogie simplă cu modul în care este privită o scenă reală prin ochiul uman sau printr-un aparat fotografic (cameră). Poziția şi orientarea sistemului de referință de observare corespund poziției şi orientării ochiului sau camerei într-o scenă reală. Dintr-un punct de observare dat, numai o parte a scenei poate fi văzută, restul aflându-se în spatele sau în părțile laterale ale direcției de observare. Partea vizibilă din scenă (reală sau virtuală) este definită ca volum de vizualizare (*viewing volume*), care, în general, este o piramidă (sau un trunchi de piramidă) cu centrul în punctul de observare şi cu direcția dată de direcția de observare. În redarea scenelor, deschiderea unghiulară a piramidei de vizualizare defineşte câmpul de vizualizare (*field of view*), care este o caracteristică importantă a sistemelor grafice.

Deşi există şi unele dispozitive de afişare tridimensională, majoritatea sistemelor grafice afişează imaginile pe o suprafață plană a unui dispozitiv de afişare (display). De aceea, în succesiunea operațiilor de vizualizare a obiectelor sau a scenelor tridimensionale, este necesară introducerea unei proiecții, care converteşte spațiul tridimensional într-o reprezentare bidimensională. În general, proiecțiile transformă spațiul R^n în spațiul R^m, unde $m < n$.

Modul în care se realizează proiecția obiectelor tridimensionale se defineşte în sistemul de referință de observare prin mai mulți parametri, dintre care cei mai importanți sunt: tipul proiecției şi planul de vizualizare (*view plane*). Intersecția dintre planul de vizualizare şi piramida de vizualizare defineşte o regiune plană numită fereastră de vizualizare (*view plane window*), în care sunt reprezentate toate imaginile bidimensionale ale obiectelor (sau părților de obiecte) vizibile din scenă.

În grafica tridimensională există mai multe posibilități de definire a sistemelor de vizualizare, de la definiri simple, utile în mici programe grafice, până la definiri complete, care permit crearea simultană a mai multor imagini ale scenei, în ferestre de vizualizare diferite. În majoritatea acestor sisteme de vizualizare, din motive de eficiență şi precizie a calculelor (care sunt explicate în continuare),

transformarea de proiecţie se execută în două etape: prima etapă este o transformare de normalizare, care este o schimbare a sistemului de referinţă de la sistemul de referinţă de observare la sistemul de referinţă normalizat. Volumul de vizualizare definit în sistemul de referinţă de observare este transformat, în sistemul de referinţă normalizat, într-un volum canonic (în general, un paralelipiped dreptunghic).

În sistemul de referinţă normalizat (*normalized coordinates*) se execută decuparea obiectelor (*clipping*), astfel încât obiectele sau părţile din obiecte care se află în afara volumului de vizualizare (volumul canonic) sunt eliminate (ignorate pentru operaţiile următoare de redare, nu şterse din scenă).

Următoarea transformare geometrică este transformarea din sistemul normalizat în *sistemul de referinţă ecran tridimensional* (*three-dimensional sceen*), prin care fereastra definită în planul de vizualizare este transformată într-o regiune corespunzătoare zonei de afişare pe display, numită poartă (*viewport*), iar coordonata z se păstrează nemodificată. În sistemul de referinţă ecran 3D se calculează coordonatele tridimensionale ale punctelor, din coordonatele omogene, prin împărţirea cu w, după care se execută transformarea de rastru (*rasterization*). Transformarea de rastru calculează mulţimea pixelilor care aparţin unui segment sau poligon dat prin coordonatele vârfurilor. Prin această transformare (care se mai numeşte şi conversie de baleiere – *scan conversion*) spaţiul bidimensional continuu al porţii de afişare este convertit în spaţiul bidimensional discret al imaginii. În general, concomitent cu transformarea de rastru sunt executate şi alte operaţii: eliminarea suprafeţelor ascunse (*hidden surface removal*), umbrirea (*shading*), texturarea (*texturing*).

Aceasta este succesiunea cea mai generală a operaţiilor de vizualizare a scenelor, care este descrisă şi utilizată în momentul de faţă în grafică şi realitatea virtuală. Trebuie remarcat faptul că evoluţia tehnologică, deosebit de spectaculoasă în acest domeniu, a modificat, uneori substanţial, modul de abordare a generării imaginii obiectelor tridimensionale. De exemplu, posibilitatea de implementare eficientă (uneori chiar în hardware) a operaţiei de comparare a adâncimii punctelor a impus utilizarea algoritmului cunoscut sub numele de algoritm Z-buffer şi a condus la ignorarea completă a altor algoritmi de eliminare a suprafeţelor ascunse, care în sistemele mai vechi aveau o mare importanţă. Se pot da numeroase exemple de modul în care schimbările tehnologice au influenţat abordarea graficii tridimensionale, şi, probabil, astfel de influenţe vor continua să apară şi în viitor. De aceea, lucrarea îşi propune să prezinte aspectele esenţiale ale graficii utilizate în realitate virtuală, mai puţin influenţate de contextul tehnologic de moment.

4.1 TRANSFORMAREA DE OBSERVARE

Transformarea de observare este analogă cu poziţionarea unei camere fotografice sau de înregistrare, prin care se observă o anumită zonă din scena virtuală. În sistemul de referinţă universal sunt reprezentate toate obiectele

(modelele) scenei virtuale, iar în fiecare cadru al imaginii (*frame*), este generată imaginea acelor obiecte din scenă care sunt vizibile din punctul de observare curent (*view point*).

Transformarea de observare este, aşadar, o schimbare a sistemului de coordonate de la sistemul de referinţă universal la sistemul de referinţă de observare (*view coordinate*). În majoritatea aplicaţiilor de realitate virtuală (simulatoare, aplicaţii în arhitectură, etc) scena virtuală reprezentată în sistemul de referinţă universal este menţinută într-o poziţie constantă (cel mult existând anumite obiecte mobile în scenă), iar observatorul este cel care îşi modifică poziţia şi orientarea, şi în funcţie de acestea se redă imaginea corespunzătoare pe ecran. Direcţia de observare a scenei este, în majoritatea sistemelor de vizualizare, direcţia axei z a sistemului de referinţă de observare.

Localizarea şi orientarea sistemului de referinţă de observare $O_V x_V y_V z_V$ în raport cu sistemul de referinţă universal Oxyz poate fi descrisă printr-o matrice de transformare **M** de forma generală:

$$\mathbf{M}=\mathbf{T_V R_V}=\begin{bmatrix}1&0&0&x_V\\0&1&0&y_V\\0&0&1&z_V\\0&0&0&1\end{bmatrix}\begin{bmatrix}c_{11}&c_{21}&c_{31}&0\\c_{12}&c_{22}&c_{32}&0\\c_{13}&c_{23}&c_{33}&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}c_{11}&c_{21}&c_{31}&x_V\\c_{12}&c_{22}&c_{32}&y_V\\c_{13}&c_{23}&c_{33}&z_V\\0&0&0&1\end{bmatrix} \quad (4.1)$$

Matricea componentă $\mathbf{T_V}$ este matricea de translaţie a sistemului de referinţă de observare în raport cu sistemul de referinţă universal: x_V, y_V, z_V sunt coordonatele punctului de observare O_V în sistemul Oxzy. Matricea de rotaţie $\mathbf{R_V}$ reprezintă orientarea sistemului de referinţă de observare faţă de sistemul de referinţă universal: coeficienţii c_{11}, c_{12}, c_{13} sunt cosinuşii directori ai axei $O_V x_V$ faţă de axele Ox, Oy, respectiv Oz; coeficienţii c_{21}, c_{22}, c_{23} sunt cosinuşii directori ai axei $O_V y_V$ faţă de axele Ox, Oy, respectiv Oz; coeficienţii c_{31}, c_{32}, c_{33} sunt cosinuşii directori ai axei $O_V z_V$ faţă de axele Ox, Oy, respectiv Oz.

Fiind dată matricea de poziţionare şi orientare a sistemului de referinţă observator relativ la sistemul de referinţă universal **M**, un punct oarecare P(x,z,y) din sistemul Oxyz se transformă în punctul $P_V(x_V,y_V,z_V)$ în sistemul de referinţă observator prin înmulţire cu matricea inversă, deci:

$$\mathbf{P_V'}=\mathbf{M^{-1}P}=\mathbf{R_V^{-1}T_V^{-1}P}=\mathbf{M_V P} \quad (4.2)$$

unde matricea de transformare de observare $\mathbf{M_V}$ este inversa matricei de localizare a sistemului de referinţă de observare relativ la sistemul de referinţă universal, adică $\mathbf{M_V}=\mathbf{M^{-1}}$.

Matricea de rotaţie mai poate fi precizată şi printr-o succesiune de rotaţii în raport cu axele sistemului de referinţă universal. Cea mai obişnuită convenţie pentru ordinea de specificare a rotaţiilor este: ruliu cu unghiul ρ (după axa z), tangaj cu unghiul θ (după axa x) şi giraţie cu unghiul χ (faţă de axa y). În această situaţie:

$$\mathbf{R_V}=\mathbf{R_Y}(\chi)\mathbf{R_X}(\theta)\mathbf{R_Z}(\rho) \quad (4.3)$$

În aceste relații de transformare de observare s-a presupus că sistemul de referință de observare este tot un sistem drept, la fel ca și sistemul de referință universal. Există însă și sisteme de vizualizare în care se definește sistemul de referință de observare ca un sistem orientat după regula mâinii stângi. Pentru schimbarea sistemului de referință de observare drept în sistem de referință stâng, se adaugă o transformare de oglindire (simetrie), de exemplu cu matricea $\mathbf{S}_{XY}$, iar matricea de transformare de observare devine $\mathbf{M'}_V$, unde:

$$\mathbf{M'}_V = \mathbf{S}_{XY}\mathbf{M}_V$$

Dat fiind că nu există o predilecție certă pentru una din aceste convenții posibile (sistem de referință de observare drept sau stâng), este necesar să fie precizată de fiecare dată convenția folosită, ceea ce se va face și pe parcursul acestei lucrări.

În deducerea relațiilor (4.2) și (4.3) s-a presupus că este cunoscută matricea $\mathbf{M}$ de localizare a sistemului de referință de observare față de sistemul de referință universal și atunci matricea de transformare de observare $\mathbf{M}_V$ se calculează ca matrice inversă a acesteia. Există situații în care matricea $\mathbf{M}$ nu este cunoscută și atunci matricea de transformare de observare $\mathbf{M}_V$ (sau $\mathbf{M'}_V$) trebuie să fie calculată prin diferite transformări geometrice și raționamente. În exemplul următor se urmărește un raționament de calcul al transformării de observare, pentru o anumită cerință de observare a scenei.

■ **Exemplul 4.1**

Se poate defini un sistem de referință de observare pornind de la alte condiții decât cele prezentate anterior. De exemplu, se cere definirea matricei de transformare de observare în situația în care obiectul care trebuie să fie văzut se află în centrul sistemului de referință universal, iar punctul de observare este un punct dat în spațiul tridimensional, $E(x_E, y_E, z_E)$.

Pentru ca obiectul aflat în centrul sistemului de referință universal să fie văzut din punctul E, se definește sistemul de referință de observare drept cu centrul în punctul E, și cu axa z_V orientată pe dreapta OE (cu sensul de la O către E). Direcția de observare într-un sistem de referință drept este definită (în majoritatea sistemelor de vizualizare) în sens invers direcției axei z, deci către centrul O.

Calculele de transformare se efectuează mai ușor dacă se dau coordonatele sferice ale punctului E: distanța μ a punctului E față de origine și unghiurile θ și ϕ (fig. 4.2). Coordonatele carteziene ale punctului E pot fi calculate în funcțe de coordonatele sferice:

$$\begin{cases} x_E = \mu \sin\phi \cos\theta \\ y_E = \mu \sin\phi \cos\theta \\ z_E = \mu \cos\phi \end{cases}$$

Pentru transformarea unui obiect din sistemul de referință universal în sistemul de referință de observare astfel definit, sunt necesare trei transformări elementare. În fig. 4.2 sunt reprezentate primele două transformări componente ale transformării de observare.

Translația $\mathbf{T_1} = \mathbf{T}(-x_E, -y_E, -z_E)$ transformă sistemul de referință universal într-un sistem de referință cu centrul în punctul E şi cu axele x', y', z' paralele cu axele sistemului de referință universal.

$$\mathbf{T_1} = \begin{bmatrix} 1 & 0 & 0 & -\mu\cos\theta\ \sin\phi \\ 0 & 1 & 0 & -\mu\sin\theta\sin\phi \\ 0 & 0 & 1 & -\mu\cos\phi \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

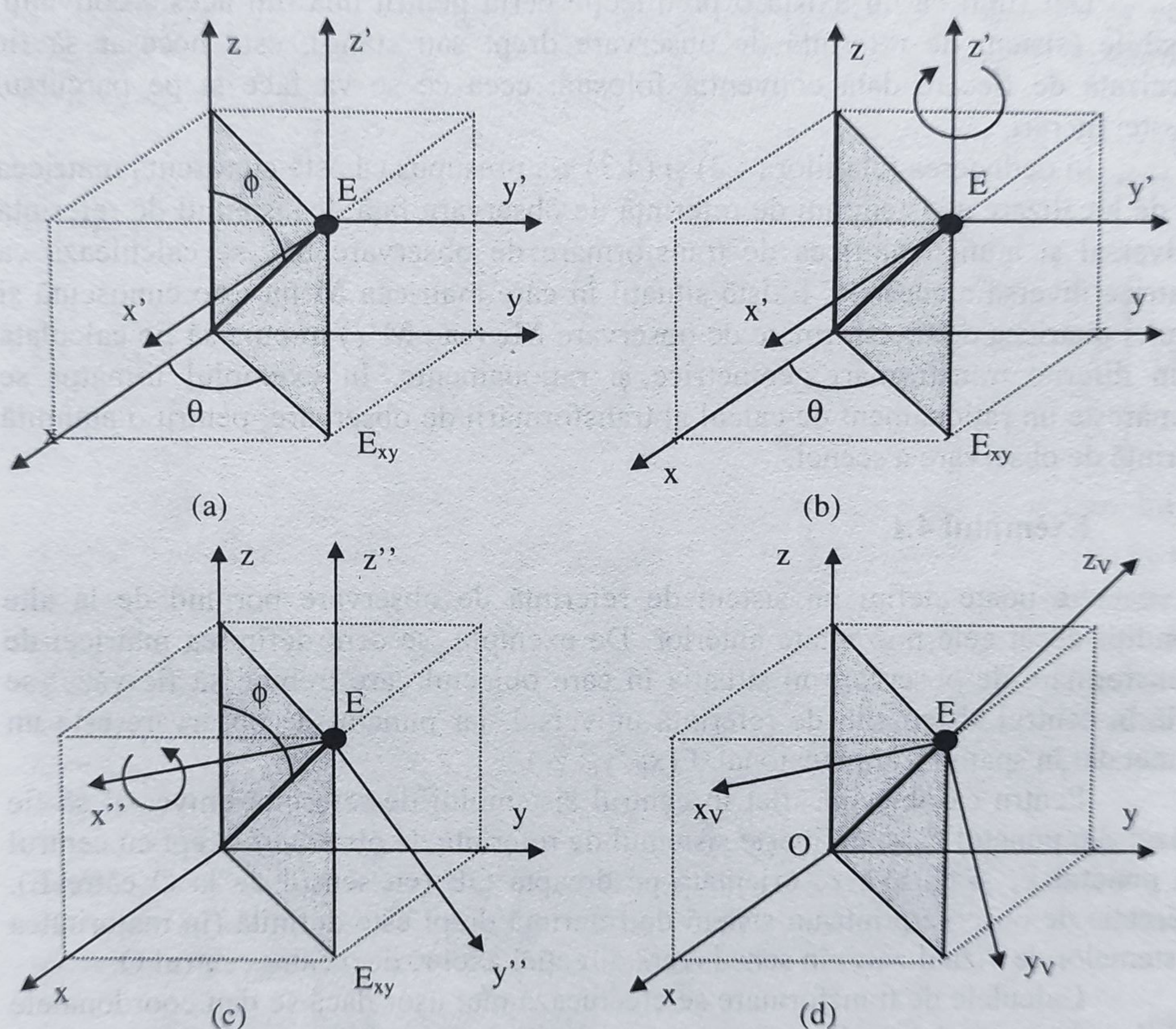


Fig. 4.2 Transformarea de observare: (a) translația (b) rotația în raport cu axa z' (c) rotația în raport cu axa x''(d) sistemul de referință rezultat.

Orientarea sistemului de referință de observare astfel încât axa z' să se suprapună peste dreapta OE se obține prin două rotații.

Prima rotație se efectuează în raport cu axa z' cu un unghi de 90° – θ în sens negativ, astfel încât noua axă x'' obținută să fie normală la planul care trece prin axa z şi punctul E (planul OEE_{xy}) (fig. 4.2(b)). Această operație este echivalentă cu o rotație în sens direct aplicată asupra punctelor, deci cu matricea $\mathbf{R_2} = \mathbf{R_Z}(90° - \theta)$, unde:

$$\mathbf{R}_2 = \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

După această rotație, sistemul de referință de observare are axa x'' perpendiculară pe planul planul OEE_{xy} şi axa Ez'' în planul OEE_{xy}.

A doua rotație este o rotație a sistemului de coordonate Ex''y''z'' în sens negativ cu un unghi egal cu ϕ în raport cu axa Ex'', pentru a suprapune axa Ez'' peste dreapta OE (fig. 4.2(a)). Acestă rotație este echivalentă cu o rotație inversă aplicată obiectelor, deci cu matricea $\mathbf{R}_3 = \mathbf{R}_X(\phi)$:

$$\mathbf{R}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rezultă matricea de transformare de observare:

$$\mathbf{M}_V = \mathbf{R}_X(\phi)\mathbf{R}_Z(90^\circ - \theta)\mathbf{T}(-x_E, -y_E, -z_E)$$

$$\mathbf{M}_V = \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta\cos\phi & \sin\theta\cos\phi & -\sin\phi & 0 \\ \cos\theta\sin\phi & \sin\theta\sin\phi & \cos\phi & -\mu \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformarea unui punct P(x,y,z) din sistemul de referință universal în punctul $P(x_V, y_V, z_V)$ în sistemul de referință de observare se obține astfel:

$$[x_V \quad y_V \quad z_V \quad 1]^T = \mathbf{M}_V [x \quad y \quad z \quad 1]^T$$

În fig. 4.3 este reprezentată imaginea unui cub cu centrul în origine şi laturile egale cu 2 şi paralele cu axele de coordonate, observat din punctul E(10,10,10). Pentru punctul E:

$$\mu = 10\sqrt{3}$$

$$\cos\theta = \sqrt{2}/2, \sin\theta = \sqrt{2}/2, \theta = 45^\circ$$

$$\cos\phi = \sqrt{3}/3, \sin\phi = \sqrt{6}/3, \phi = 54^\circ 45'$$

Matricea de transformare de observare are valoarea:

$$\mathbf{M}_V = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{6}/6 & \sqrt{6}/6 & -\sqrt{6}/3 & 0 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 & -10\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

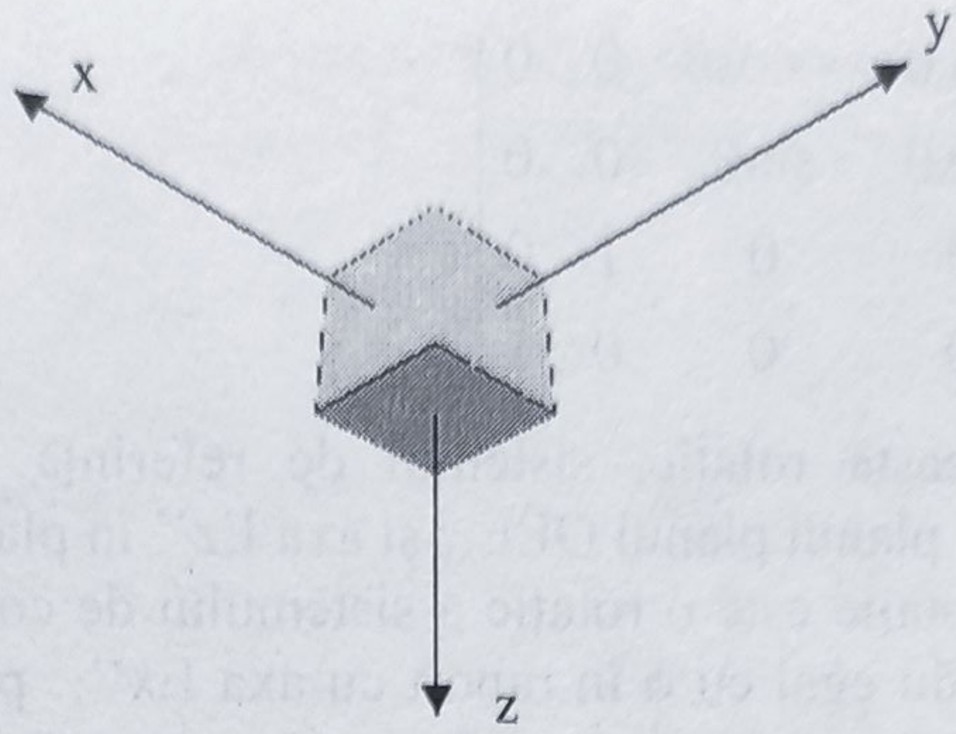


Fig. 4.3 Exemplu de observare către centrul sistemului de referință universal.

Centrul O al sistemului de referință universal se transformă în punctul $O'(0,0,-9\sqrt{3})$ în sistemul de referință de observare, iar vârful cubului P(1,1,1), se transformă în punctul $P'(0,0,-10\sqrt{3})$. Direcția de observare trece printr-un vârf al cubului şi prin centrul lui.

4.2 TRANSFORMAREA DE PROIECȚIE

Proiecția unui obiect tridimensional pe un plan (numit plan de proiecție sau plan de vizualizare – *view plane*) este formată din mulțimea punctelor de intersecție între acest plan şi dreptele numite proiectori, care pornesc dintr-un punct fix numit centru de proiecție şi trec prin fiecare punct al obiectului.

În funcție de distanța centrului de proiecție față de planul de proiecție, proiecțiile plane se clasifică în proiecții perspectivă şi proiecții paralele. În proiecțiile perspectivă centrul de proiecție se află la distanță finită față de planul de vizualizare, iar în proiecțiile paralele centrul de proiecție este la infinit. În cazul proiecțiilor paralele, proiectorii sunt drepte paralele şi au o direcție comună, numită direcție de proiecție.

În grafica tridimensională se foloseşte cel mai frecvent proiecția perspectivă, deoarece această proiecție redă cel mai bine efectul de descreştere a dimensiunilor obiectelor odată cu creşterea distanței acestora față de observator.

Modul de definire a punctului de observare a scenei, a centrului de proiecție şi a planului de vizualizare depind de sistemul de vizualizare folosit. Un studiu atent al sistemelor de vizualizare este necesar din mai multe motive. Chiar dacă un utilizator nu trebuie să proiecteze el însuşi sistemul de vizualizare, exploatarea oricărui sistem necesită o înțelegere precisă a modului în care diferiți parametri determină imaginea obținută pe display. Combinațiile posibile sunt în număr destul de mare, astfel încât şansa de a adecva diferiți parametri ai unui sistem de vizualizare, fără a fi cunoscute condițiile în care lucrează acesta, este foarte redusă.

4.2.1 PROIECŢIA PARALELĂ

În proiecţia paralelă, centrul de proiecţie este la infinit, iar toate dreptele de proiecţie sunt paralele între ele, având o direcţie numită direcţia de proiecţie.

Proiecţiile paralele sunt foarte rar folosite în grafica destinată sistemelor de realitate virtuală, dat fiind că imaginea rezultată nu dă informaţii asupra distanţei obiectelor în scenă.

Planul de proiecţie al unei proiecţii paralele poate fi perpendicular pe direcţia de proiecţie, şi atunci proiecţia se numeşte *proiecţie ortografică*, sau poate avea o înclinare oarecare faţă de direcţia de proiecţie, şi atunci proiecţia se numeşte *proiecţie oblică*.

Dintre proiecţiile ortografice, cele mai utilizate sunt proiecţiile ortografice în care planul de proiecţie este unul din planurile sistemului de coordonate. Proiecţiile ortografice sunt folosite frecvent în desenul tehnic şi, de asemenea, ca o etapă intermediară în definirea sistemelor de vizualizare normalizate, care vor fi prezentate în paragraful următor. Matricea de transformare de proiecţie ortografică în planul z = 0 are următoarea expresie:

$$\mathbf{P_Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Prin proiecţie ortografică în planul z = 0, un punct P(x,y,z) se transformă în punctul P'(x,y,0), care are aceleaşi coordonate x şi y, iar z are valoarea zero.

În mod similar, se definesc matricele de proiecţie ortografică în planele x = 0 şi y = 0:

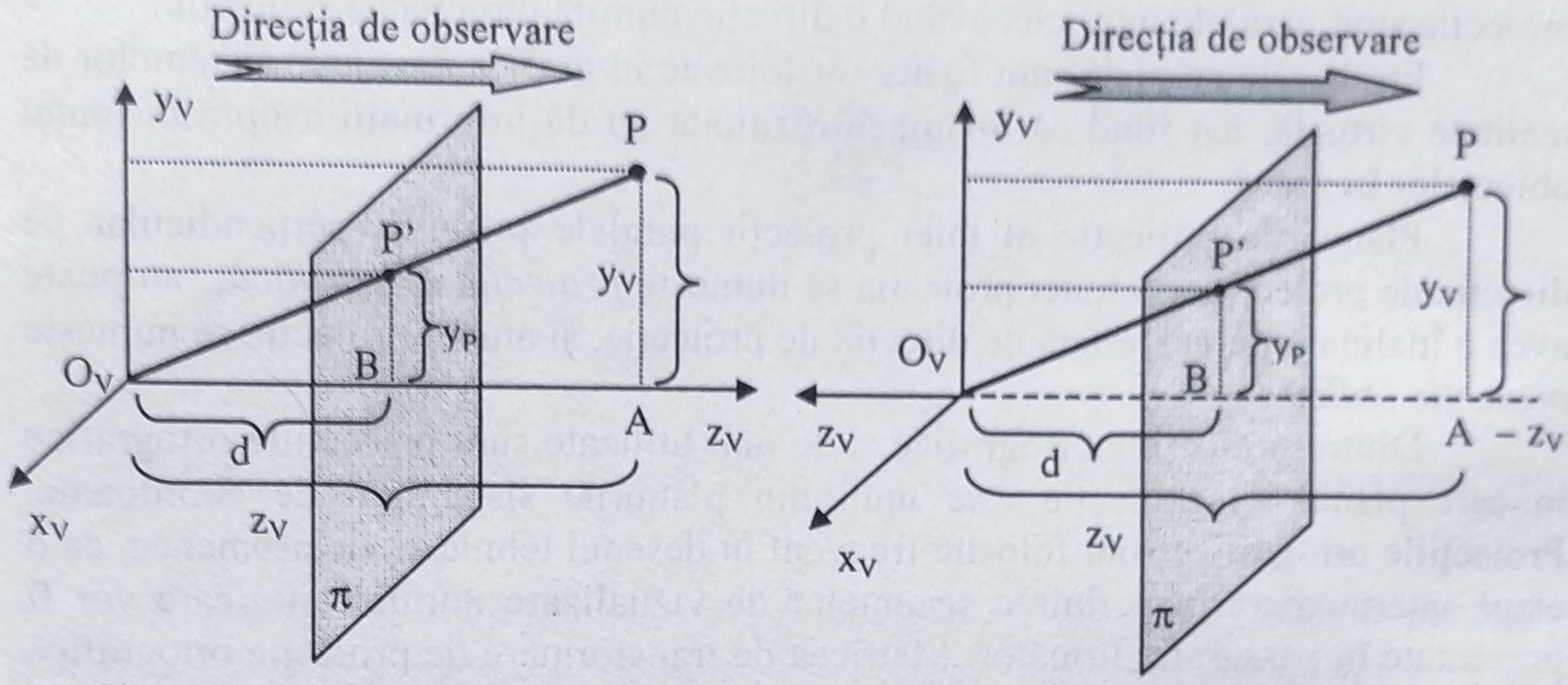
$$\mathbf{P_X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P_Y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

4.2.2 PROIECŢIA PERSPECTIVĂ

Există mai multe modalităţi de definire a unei proiecţii perspectivă, de la o definire minimală, prin care se poate realiza proiecţia punctelor din spaţiu pe un plan, până la definirea adoptată în sistemele de vizualizare complexe, în care se specifică volumul de vizualizare, centrul de proiecţie şi alţi parametri. În continuare, vor fi prezentate câteva sisteme de vizualizare cu proiecţie perspectivă, începând cu cel mai simplu sistem şi până la sistemul complet de vizualizare care se conformează standardelor GKS şi PHIGS.

Proiecţia perspectivă se poate defini mai intuitiv pornind de la sistemul de observare considerat ca un sistem de referinţă stâng. Centrul de proiecţie se fixează în centrul sistemului de referinţă de observare O_V, direcţia de observare (de privire)

este în sensul pozitiv al axei O_Vz_V, iar planul de vizualizare π este un plan perpendicular pe axa O_Vz_V, la distanța d față de centru (fig. 4.4(a)).



(a) Sistem de referință de observare stâng (b) Sistem de referință de observare drept

Fig. 4.4 Proiecția perspectivă definită prin centrul de proiecție O_V și planul de proiecție π perpendicular pe axa O_Vz_V.

Punctul $P(x_V,y_V,z_V)$ în sistemul de referință de observare se proiectează în punctul $P'(x_P,y_P,z_P)$, aflat la intersecția dreptei O_VP cu planul de vizualizare. Din triunghiurile asemenea $O_VP'A$ şi O_VPB, rezultă:

$$y_P = \frac{d\, y_V}{z_V}$$

În mod asemănător se poate deduce valoarea lui x_P:

$$x_P = \frac{d\, x_V}{z_V} \tag{4.6}$$

Aceste relații se pot exprima matriceal prin definirea matricei de proiecție perspectivă $\mathbf{M_P}$ astfel:

$$\mathbf{M_P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{4.7}$$

Transformarea de proiecție se poate scrie atunci:

$$\mathbf{P'} = \mathbf{M_P P}, \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ w_P \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} X_V \\ Y_V \\ Z_V \\ w_V \end{bmatrix}$$

Rezultă:

$$\begin{cases} X_P = X_V \\ Y_P = Y_V \\ Z_P = Z_V \\ w_P = Z_V / d \end{cases} \text{ şi } \begin{cases} x_P = X_P / w_P = d\, X_V / Z_V = d\, x_V / z_V \\ y_P = Y_P / w_P = d\, Y_V / Z_V = d\, y_V / z_V \\ z_P = Z_P / w_P = d \end{cases} \quad (4.8)$$

Se observă că expresiile lui x_P, y_P şi z_P obţinute prin calcul matriceal sunt identice cu cele din relaţiile (4.6), rezultate din construcţia geometrică prezentată în fig 4.4(a). Proiecţia perspectivă necesită deci o împărţire a coordonatelor x şi y cu distanţa z, şi, într-adevăr, în proiecţia perspectivă imaginea unui obiect este cu atât mai mică cu cît se află la distanţă mai mare de punctul de observare.

În acest sistem de vizualizare simplu s-a definit sistemul de referinţă de observare ca un sistem stâng. Se poate defini acest sistem ca un sistem de referinţă drept, iar direcţia de observare îndreptată spre $-z_V$. Planul de vizualizare este poziţionat perpendicular pe axa z, la distanţă egală cu d faţă de origine (fig. 4.4(b)), d având valoare negativă. Urmând un raţionament asemănător cu cel de mai înainte, rezultă matricea de proiecţie perspectivă cu aceeaşi expresie (4.7), în care d are o valoare negativă. Coordonatele punctului transformat sunt date de relaţiile (4.8).

Se remarcă echivalenţa rezultatelor obţinute, iar convenţia de sistem de referinţă de observare stâng este utilă doar pentru a se urmări mai uşor raţionamentul de calcul.

4.2.3 SISTEMUL DE REFERINŢĂ NORMALIZAT

Sistemul de proiecţie simplu definit anterior, în care toate obiectele din scenă sunt proiectate pe planul de vizualizare, are mai multe inconveniente şi nu poate fi folosit decât în aplicaţii foarte simple, în care se proiectează obiecte aflate tot timpul în câmpul de observare. Nu aceasta este situaţia imaginilor generate în realitatea virtuală în care se explorează o scenă virtuală de dimensiuni mari, din care numai o parte este vizibilă şi deci redată pe ecran.

Transformarea de proiecţie perspectivă este o transformare costisitoare, datorită faptului că necesită operaţii de împărţire, care consumă timp de calcul ridicat, dacă este executată prin program, sau circuite complexe, dacă este implementată hardware. Execuţia ei pentru toate obiectele scenei, inclusiv pentru cele care nu sunt vizibile, deci nu sunt reprezentate pe display şi nu necesită coordonatele în planul de vizualizare, este costisitoare şi inutilă. În plus, operaţia de decupare (*clipping*) a părţilor din obiectele tridimensionale care nu sunt vizibile dintr-un anumit punct de observare nu se poate executa eficient după proiecţie şi este necesară execuţia ei înainte de proiectarea coordonatelor în planul de vizualizare. Asupra acestui aspect se va reveni în § 4.5.2.

Este evident că, pentru redarea eficientă a scenelor tridimensionale, este necesară definirea unui volum de vizualizare (*view volume*), cunoscut sub numele de piramida de vizualizare sau trunchiul de piramidă de vizualizare (*frustum*).

Trunchiul de piramidă de vizualizare se defineşte în sistemul de referinţă de observare. În forma cea mai simplă, trunchiul de piramidă de vizualizare este mărginit de patru plane laterale care trec prin punctul de observare O_V şi de două plane perpendiculare pe axa O_Vz_V, planul apropiat (*near*) pe care se execută proiecţia, şi planul depărtat (*far*). În fig. 4.5 este prezentat volumul de vizualizare într-un sistem de referinţă de observare stâng şi direcţia de observare către z_V pozitiv.

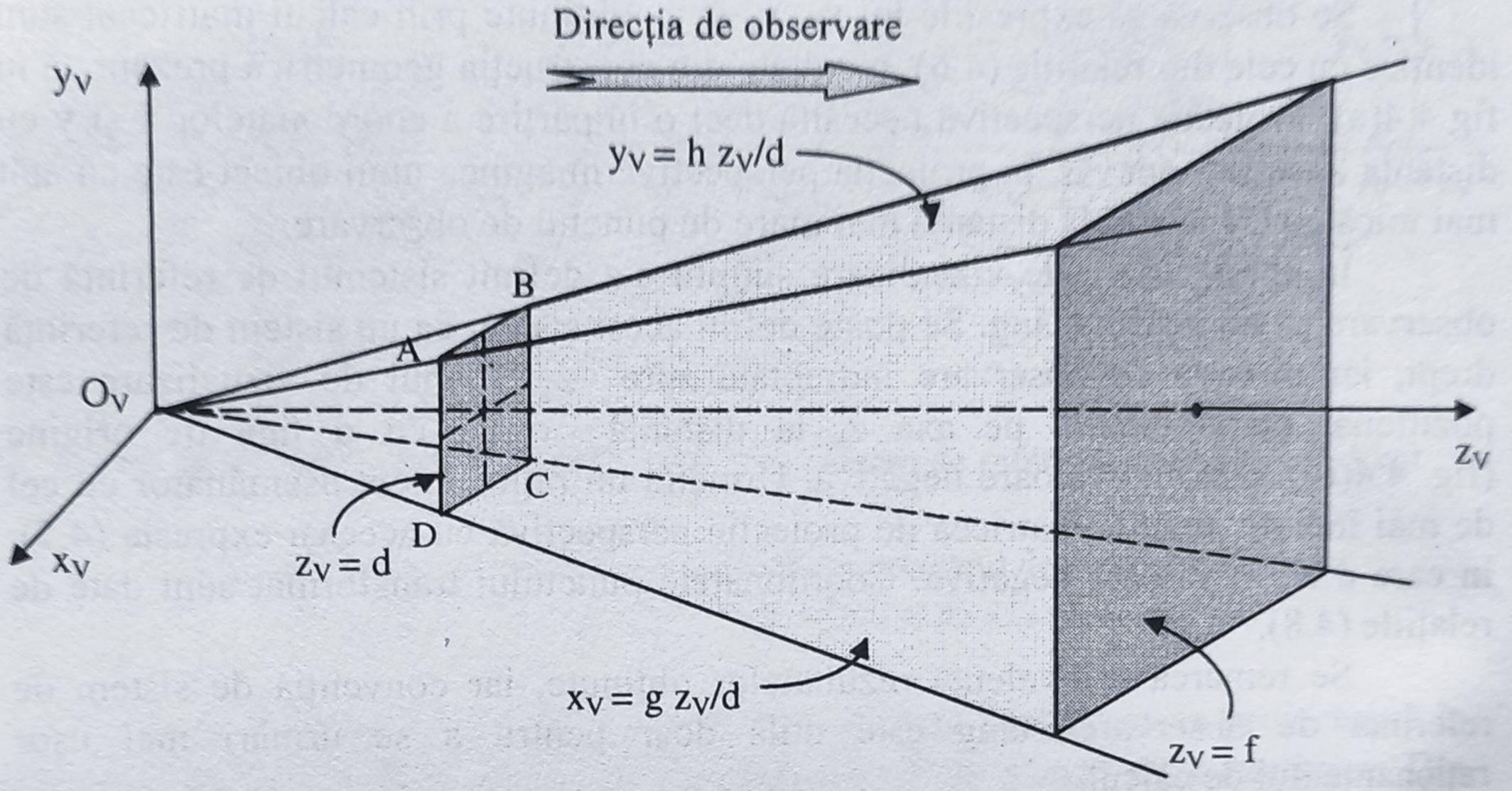


Fig. 4.5 Definirea trunchiului de piramidă de vizualizare (frustum) în sistemul de referinţă de observare stâng.

Intersecţia dintre cele patru plane laterale şi planul de vizualizare (care, în această definiţie, este şi planul apropiat de vizibilitate) determină o suprafaţă dreptunghiulară ABCD în planul de vizualizare în care vor fi proiectate toate obiectele vizibile ale scenei. Această suprafaţă este fereastra de vizualizare (*view plane window*). Ecuaţiile celor şase plane ale trunchiului de piramidă regulată sunt:

$$\begin{cases} x_V = \pm g\, z_V / d \\ y_V = \pm h\, z_V / d \\ z_V = d \\ z_V = f \end{cases} \tag{4.9}$$

unde: $2g$ – dimensiunea laturii orizontale a ferestrei de vizualizare (latura AB)
$2h$ – dimensiunea laturii verticale a ferestrei de vizualizare (latura BC)
d – distanţa planului de vizualizare şi a planului apropiat
f – distanţa planului depărtat.

Semnificaţia volumului de vizualizare este evidentă: dintre toate obiectele din scena virtuală sunt vizibile şi redate în fereastra de vizualizare numai acele obiecte sau părţi din obiecte care sunt cuprinse în acest volum. Trunchiul de

piramidă de vizualizare defineşte în acest fel şi volumul de decupare, operaţie care se execută într-unul din sistemele de referinţă care se definesc în cadrul transformării de proiecţie, şi anume sistemul de referinţă normalizat. Acest sistem de referinţă se mai numeşte şi sistem de referinţă canonic. Motivele pentru care decuparea se efectuează cel mai eficient în sistemul de referinţă normalizat vor fi explicate ulterior.

Sistemul de referinţă normalizat este un sistem de coordonate tridimensional în care trunchiul de piramidă de vizualizare se transformă într-un volum canonic, care, în general, este un paralelipiped dreptunghic. Pentru sistemul de proiecţie definit în acest paragraf, *volumul canonic* este un paralelipiped dreptunghic cu latura bazei egală cu 2 şi înălţimea 1 (după axa z_N).

Introducerea acestui nou sistem de referinţă permite abordarea transformării de proiecţie perspectivă în două etape: o transformare din sistemul de referinţă de observare în sistemul de referinţă normalizat (numită transformare de normalizare), urmată de calculul proiecţiei prin împărţirea cu w a coordonatelor omogene. Între cele două etape ale transformării de proiecţie perspectivă se execută o operaţie importantă în grafica tridimensională, decuparea obiectelor relativ la volumul de vizualizare normalizat (volumul canonic).

Pentru transformarea de normalizare (transformarea de la sistemul de referinţă de observare la sistemul de referinţă normalizat), se impun următoarele cerinţe:

(1) Se normalizează coordonata z_N, care reprezintă adâncimea, astfel încât să se obţină precizie maximă în operaţiile de comparaţie a adâncimii punctelor.
(2) Liniile din sistemul de referinţă observator trebuie să se transforme în linii în sistemul de referinţă normalizat.
(3) Suprafeţele plane din sistemul de referinţă observator trebuie să se transforme în suprafeţe plane în sistemul de referinţă normalizat.

S-a demonstrat [New81] că aceste cerinţe sunt satisfăcute de transformarea:

$$z_N = a + b/z_V$$

unde a şi b sunt constante. Aceste constante se determină din următoarele condiţii:

- Se alege $b < 0$, astfel ca z_N să crească atunci când creşte z_V. Această condiţie conservă noţiunea intuitivă de adâncime: un punct are coordonata z_N cu atât mai mare cu cât este mai departe de observator.
- Se normalizează domeniul de variaţie al lui z_N, astfel ca domeniul $z_V \in [d,f]$ să se transforme în domeniul $z_N \in [0,1]$.

Rezultă următoarele relaţii care definesc transformarea de normalizare:

$$x_N = \frac{d\, x_V}{g\, z_V}; \quad y_N = \frac{d\, y_V}{h\, z_V}; \quad z_N = \frac{f}{f-d}\left(1 - \frac{d}{z_V}\right) \tag{4.10}$$

Coordonatele omogene ale unui punct în sistemul normalizat au expresiile:

$$\begin{cases} X_N = d\,x_V / g \\ Y_N = d\,y_V / h \\ Z_N = z_V f /(f-d) - df/(f-d) \\ w_N = z_V \end{cases} \tag{4.11}$$

Din aceste expresii se deduce matricea de transformare de normalizare $\mathbf{M_N}$ care efectuează transformarea de la sistemul de referință observator la sistemul de referință normalizat:

$$\mathbf{M_N} = \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix} = \mathbf{M_N} \begin{bmatrix} x_V \\ y_V \\ z_V \\ 1 \end{bmatrix} \tag{4.12}$$

Matricea de transformare de normalizare $\mathbf{M_N}$ poate fi exprimată ca un produs de două matrice astfel:

$$\mathbf{M_N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{M_{N2}} \mathbf{M_{N1}}$$

Această reprezentare permite analiza mai detaliată a transformării de normalizare. Sistemul de referință normalizat astfel definit este un sistem de referință stâng. Matricea $\mathbf{M_{N1}}$ este o matrice de scalare neuniformă prin care trunchiul de piramidă de vizualizare este transformat într-un trunchi de piramidă regulată cu unghiul la vârful piramidei de 90°. Matricea $\mathbf{M_{N2}}$ transformă acest trunchi de piramidă regulată în volumul canonic, care este un paralelipiped dreptunghic (fig. 4.6).

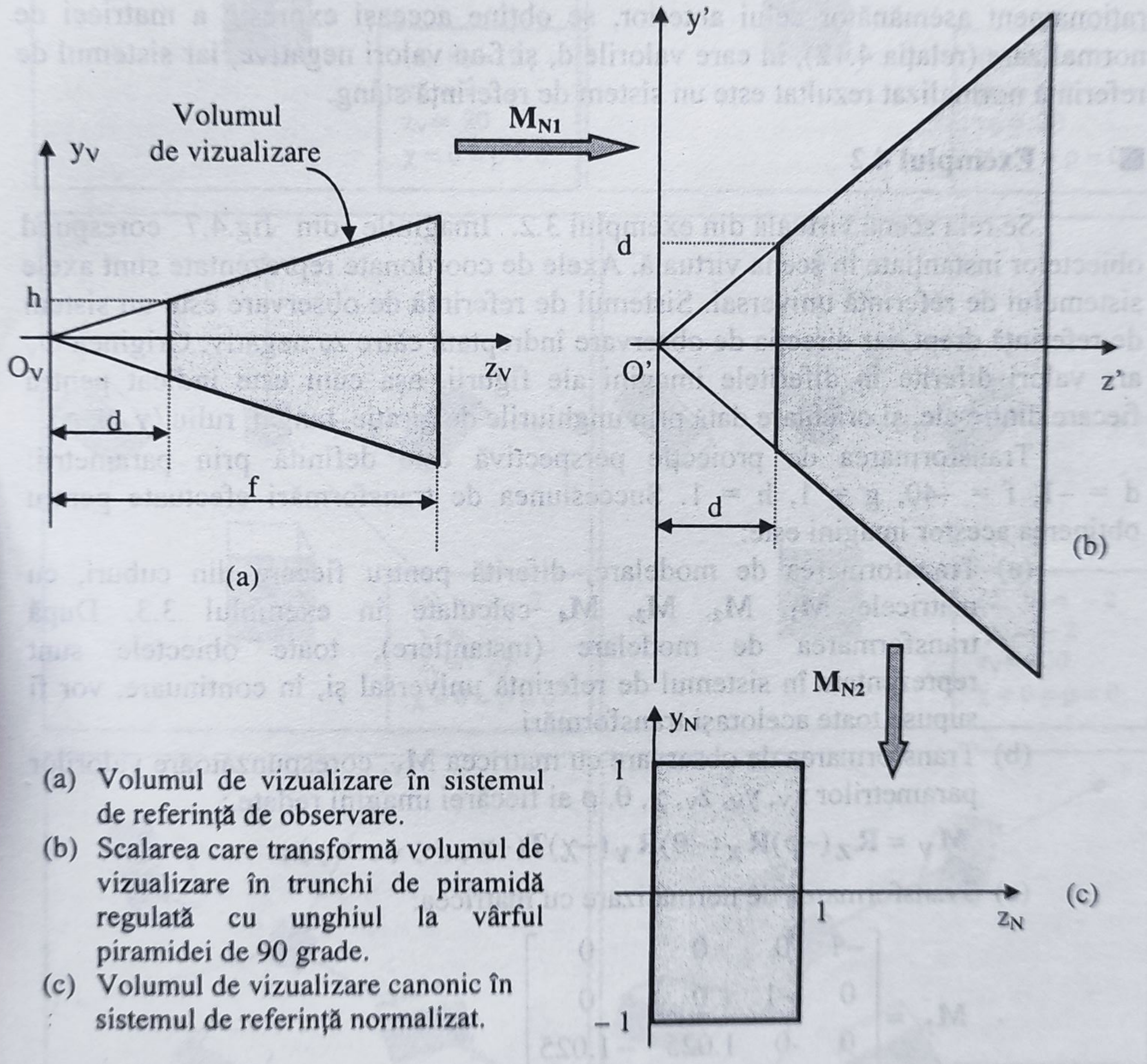
În sistemul de referință normalizat se execută operația de decupare a obiectelor în coordonate normalizate omogene. În urma acestei operații, din totalitatea obiectelor scenei virtuale sunt reținute numai acele obiecte sau părți (rezultate prin decupare) aflate în interiorul volumului de vizualizare canonic, ceea ce înseamnă că, înainte de transformarea de normalizare, acestea se aflau în interiorul trunchiului de piramidă de vizualizare.

Din coordonatele omogene X_N, Y_N, Z_N , w_N se pot deduce coordonatele x_N, y_N și z_N ale punctului în sistemul de referință normalizat tridimensional:

$$x_N = X_N / w_N = \frac{d\,x_V}{g\,z_V};\; y_N = Y_N / w_N = \frac{d\,y_V}{h\,z_V};\; z_N = Z_N / w_N = \frac{f}{f-d}\left(1 - \frac{d}{z_V}\right)$$

S-au obținut, evident, expresii identice cu cele din relațiile (4.10), de la care s-a pornit pentru definirea transformării de proiecție perspectivă.

Coordonatele x_N şi y_N în sistemul de referinţă normalizat reprezintă coordonatele proiecţiei ortografice în planul $z_N = 0$ (care este planul de vizualizare) a punctului corespunzător. Este de remarcat faptul că transformarea de proiecţie perspectivă din sistemul de referinţă de observare a devenit o transformare de proiecţie ortografică în sistemul de referinţă normalizat. Dar aceaste operaţii de trecere de la coordonate omogene la coordonate tridimensionale şi proiecţia ortografică nu se efectuează în sistemul normalizat ci într-un alt sistem de referinţă, sistemul ecran 3D, obţinut prin transformarea ferestrei de vizualizare în poartă de afişare.



(a) Volumul de vizualizare în sistemul de referinţă de observare.
(b) Scalarea care transformă volumul de vizualizare în trunchi de piramidă regulată cu unghiul la vârful piramidei de 90 grade.
(c) Volumul de vizualizare canonic în sistemul de referinţă normalizat.

Fig. 4.6 Etapele transformării de normalizare.

Unghiul dintre planele laterale ale piramidei de vizualizare (care trec prin axa O_vy_v şi au ecuaţiile $x_v = \pm g\ z_v/d$) se numeşte unghi de vizualizare pe orizontală (*horizontal field of view– fovx*). Unghiul dintre planele piramidei de vizualizare care trec prin axa O_vx_v se numeşte unghi de vizualizare pe verticală

(*vertical field of view – fovy*). Valorile acestora se deduc din parametrii de proiecție perspectivă:

$$\text{fovx} = 2\arctan\frac{g}{d}, \quad \text{fovy} = 2\arctan\frac{h}{d} \tag{4.13}$$

Valorile unghiurilor de vizualizare variază pentru diferite sisteme grafice în funcție de dispozitivele de afişare folosite.

O altă convenție de definire a unui sistem de vizualizare este aceea de a considera sistemul de referință de observare ca un sistem drept, iar direcția de observare şi volumul de vizualizare îndreptate către z_V negativ. Urmărind un raționament asemănător celui anterior, se obține aceeaşi expresie a matricei de normalizare (relația 4.12), în care valorile d, şi f au valori negative, iar sistemul de referință normalizat rezultat este un sistem de referință stâng.

■ **Exemplul 4.2**

Se reia scena virtuală din exemplul 3.2. Imaginile din fig.4.7 corespund obiectelor instanțiate în scena virtuală. Axele de coordonate reprezentate sunt axele sistemului de referință universal. Sistemul de referință de observare este un sistem de referință drept, iar direcția de observare îndreptată către z_V negativ. Originea O_V are valori diferite în diferitele imagini ale figurii, aşa cum este indicat pentru fiecare dintre ele, şi orientare dată prin unghiurile de girație, tangaj, ruliu (χ ,θ, ρ).

Transformarea de proiecție perspectivă este definită prin parametrii: d = –1, f = –40, g = 1, h = 1. Succesiunea de transformări efectuate pentru obținerea acestor imagini este:

(a) Transformarea de modelare, diferită pentru fiecare din cuburi, cu matricele $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_4}$ calculate în exemplul 3.3. După transformarea de modelare (instanțiere), toate obiectele sunt reprezentate în sistemul de referință universal şi, în continuare, vor fi supuse toate aceloraşi transformări.

(b) Transformarea de observare cu matricea $\mathbf{M_V}$, corespunzătoare valorilor parametrilor x_V, y_V, z_V, χ, θ, ρ ai fiecărei imagini redate.:

$$\mathbf{M_V} = \mathbf{R_Z}(-\rho)\mathbf{R_X}(-\theta)\mathbf{R_Y}(-\chi)\mathbf{T}(-x_V,-y_V,-z_V),$$

(c) Transformarea de normalizare cu matricea:

$$\mathbf{M_N} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1.025 & -1.025 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Imaginile generate pe ecran mai necesită şi alte transformări grafice, pe lângă cele prezentate până acum: transformarea în coordonate ecran 3D, prin care se efectuează o corespondență între fereastra de vizualizare şi zona de afişare alocată pe ecran (numită poartă – *viewport*), precum şi o transformare de rastru, prin care se generează mulțimea pixelilor afişați pe ecran. Aceste transformări vor fi descrise paragrafele următoare.

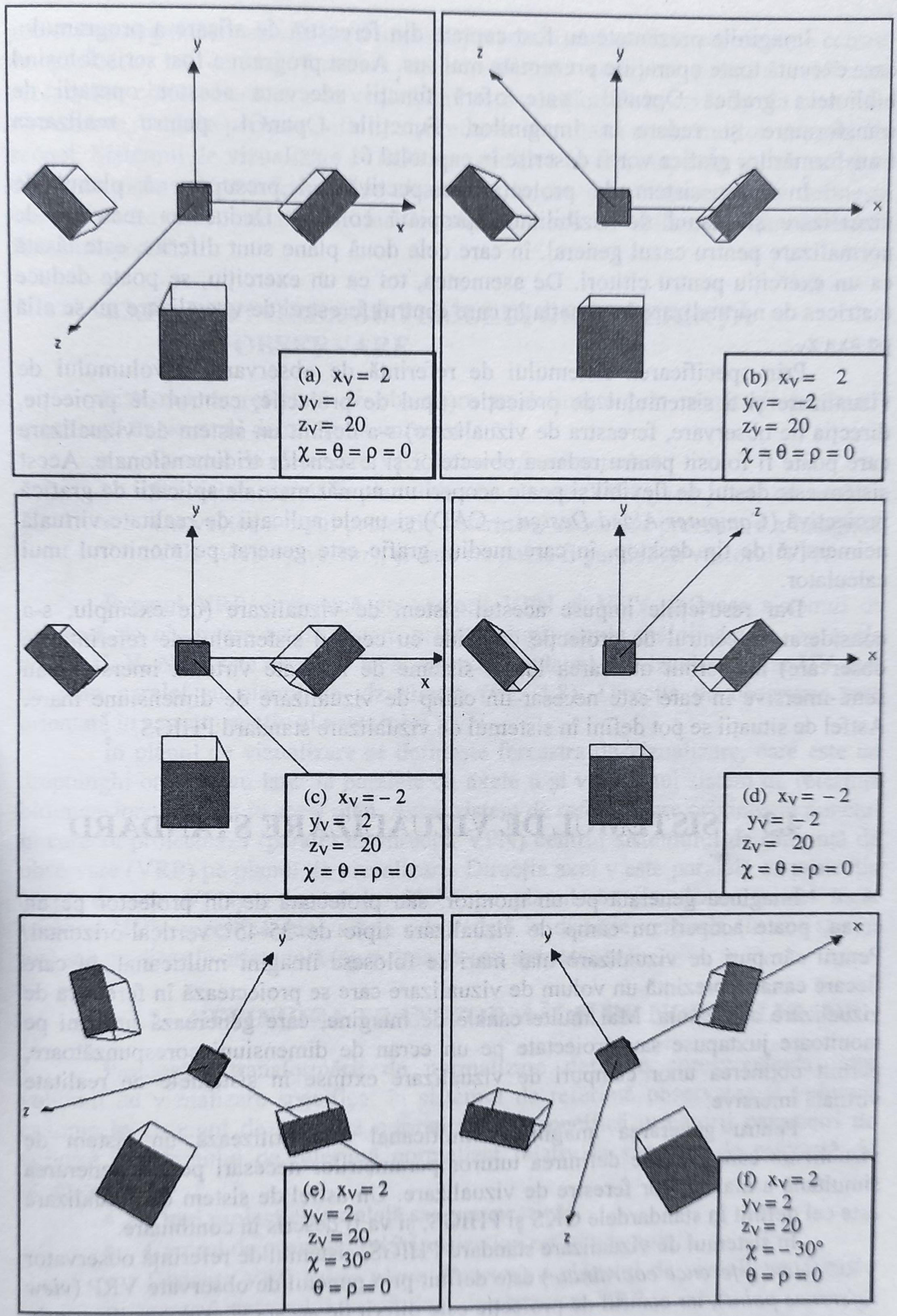


Fig. 4.7 Transformări de proiecție perspectivă.

Imaginile prezentate au fost captate din fereastra de afişare a programului care execută toate operaţiile prezentate mai sus. Acest program a fost scris folosind biblioteca grafică OpenGL, care oferă funcţii adecvate acestor operaţii de transformare şi redare a imaginilor. Funcţiile OpenGL pentru realizarea transformărilor grafice vor fi descrise în capitolul 6.

În acest sistem de proiecţie perspectivă s-a presupus că planul de vizualizare şi planul de vizibilitate apropiată coincid. Deducerea matricei de normalizare pentru cazul general, în care cele două plane sunt diferite, este lăsată ca un exerciţiu pentru cititori. De asemenea, tot ca un exerciţiu, se poate deduce matricea de normalizare în situaţia în care centrul ferestrei de vizualizare nu se află pe axa z_V.

Prin specificarea sistemului de referinţă de observare, a volumului de vizualizare şi a sistemului de proiecţie (tipul de proiecţie, centrul de proiecţie, direcţia de observare, fereastra de vizualizare) s-a definit un sistem de vizualizare care poate fi folosit pentru redarea obiectelor şi a scenelor tridimensionale. Acest sistem este destul de flexibil şi poate acoperi un număr mare de aplicaţii de grafică proiectivă (*Computer Aided Design* – CAD) şi unele aplicaţii de realitate virtuală neimersivă de tip desktop, în care mediul grafic este generat pe monitorul unui calculator.

Dar restricţiile impuse acestui sistem de vizualizare (de exemplu, s-a considerat că centrul de proiecţie coincide cu centrul sistemului de referinţă de observare) nu permit utilizarea lui în sisteme de realitate virtuală imersive sau semi-imersive în care este necesar un câmp de vizualizare de dimensiune mare. Astfel de situaţii se pot defini în sistemul de vizualizare standard PHIGS.

4.3 SISTEMUL DE VIZUALIZARE STANDARD

Imaginea generată pe un monitor, sau proiectată de un proiector pe un ecran, poate acoperi un câmp de vizualizare tipic de 35-45° vertical-orizontal. Pentru câmpuri de vizualizare mai mari se folosesc imagini multicanal, în care fiecare canal reprezintă un volum de vizualizare care se proiectează în fereastra de vizualizare a acestuia. Mai multe canale de imagine, care generează imagini pe monitoare juxtapuse sau proiectate pe un ecran de dimensiuni corespunzătoare, permit obţinerea unor câmpuri de vizualizare extinse în sistemele de realitate virtuală imersive.

Pentru generarea imaginilor multicanal se utilizează un sistem de vizualizare care permite definirea tuturor parametrilor necesari pentru generarea simultană a mai multor ferestre de vizualizare. Un astfel de sistem de vizualizare este cel definit în standardele GKS şi PHIGS, şi va fi descris în continuare.

În sistemul de vizualizare standard PHIGS, sistemul de referinţă observator VRC (*view reference coordinate*) este definit prin punctul de observare VRP (*view reference point*), iar centrul de proiecţie este diferit de acesta şi este specificat prin punctul PRP (*projection reference point*). Planul de vizualizare (pe care se execută

proiecția imaginii) nu este în mod necesar perpendicular pe linia care unește centrul de proiecție cu centrul ferestrei de vizualizare, ceea ce permite realizarea proiecțiilor oblice. Fereastra de vizualizare este poziționată în orice loc în planul de vizualizare, ceea se permite definirea şi redarea simultană a mai multor imagini ale scenei. Sistemul de vizualizare PHIGS se specifică în trei etape. În prima etapă se definește sistemul de referință de observare; în cea de-a doua etapă se definește transformarea de normalizare, iar în ultima etapă se definește transformarea fereastră-poartă.

4.3.1 DEFINIREA SISTEMULUI DE REFERINȚĂ DE OBSERVARE

Sistemul de referință de observare se definește prin următorii parametri specificați în sistemul de referință universal:

- Punctul de observare VRP (*view reference point*).
- Direcția normală la planul de vizualizare, VPN (*view plane normal*).
- Un vector a cărui proiecție determină sensul de prezentare a imaginii, VUP (*view up vector*), şi care nu poate fi paralel cu vectorul VPN.

Punctul VRP, împreună cu vectorii VPN şi VUV definesc sistemul de referință de observare ca un sistem orientat după regula mâinii drepte, cu axele notate U, V, N. N este vectorul VPN, normal la planul de vizualizare, iar UV este un plan paralel cu planul de vizualizare (fig. 4.8). Direcția de observare este orientată în sensul negativ al vectorului VPN.

În planul de vizualizare se definește fereastra de vizualizare, care este un dreptunghi orientat cu laturile paralele cu axele u şi v ale unui sistem de referință bidimensional definit în acest plan. Acest sistem de referință are originea în punctul în care se proiectează (paralel cu direcția VPN) centrul sistemului de referință de observare (VRP) pe planul de vizualizare. Direcția axei v este paralelă cu proiecția (după direcția VPN) a vectorului VUV pe planul de vizualizare, astfel încât vectorul VUV specifică dacă scena este redată în direcție verticală (în sus sau în jos) sau pe direcție orizontală (spre dreapta sau spre stânga).

4.3.2 DEFINIREA TRANSFORMĂRII DE NORMALIZARE

Parametrii transformării de normalizare definesc corespondența între volumul de vizualizare specificat în sistemul de referință observator şi volumul canonic în sistemul de referință normalizat. Se specifică următorii parametri de definire a sistemului de referință normalizat relativ la sistemul de referință de observare:

- Tipul proiecției (paralelă sau perspectivă).
- Centrul de proiecție PRP (*projection reference point*).
- Distanța VPD (*view plane distance*) a planului de vizualizare față de centrul sistemului de referință de observare VRP.

- Distanța planului apropiat (*near*) şi al planului depărtat de vizibilitate (*far*) față de centrul sistemului de referință de observare VRP.
- Limitele ferestrei de vizualizare, u_{min}, u_{max}, v_{min}, v_{max}.

Planul de vizualizare este un plan paralel cu planul UV, aflat la distanța dată de valoarea VPD față de centrul sistemului de observare VRP. În planul de vizualizare se defineşte sistemul de coordonate plan uv, în care fereastra de vizualizare este un dreptunghi între limitele u_{min}, u_{max}, v_{min}, v_{max}.Volumul de vizualizare se defineşte prin două plane paralele cu planul de vizualizare aflate la distanțele *near* şi *far* de centrul sistemului de referință de observare VRP şi planele care trec prin centrul de proiecție PRP şi laturile ferestrei de vizualizare.

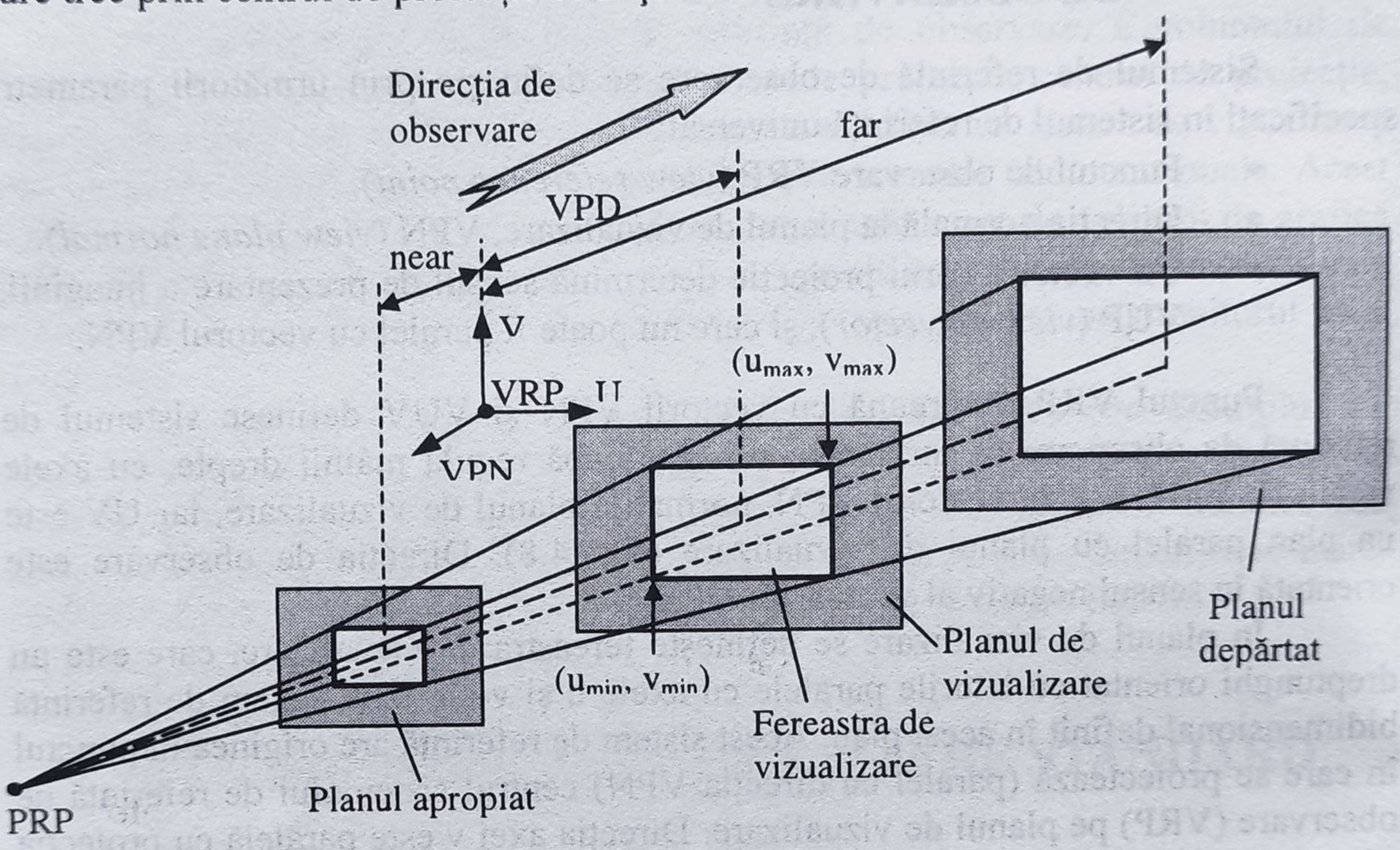


Fig. 4.8 Specificarea sistemului de vizualizare PHIGS în proiecța perspectivă.

Există două posibilități de poziționare a centrului de proiecție PRP față de fereastra de vizualizare:

(a) Dacă linia care uneşte centrul de proiecție cu centrul ferestrei de vizualizare este paralelă cu vectorul VPN, normal la planul de vizualizare, atunci se obține o proiecție perspectivă normală, echivalentă celei descrise anterior (fig. 4.9(a)).

(b) Dacă punctul PRP nu se află pe linia paralelă cu vectorul VPN care trece prin centrul ferestrei de vizualizare, atunci proiecția este o proiecție oblică (fig. 4.9(b)).

Implementarea sistemului de vizualizare PHIGS se face prin transformări geometrice succesive, la fel ca în exemplele prezentate până acum. Trebuie precizat că în sistemul PHIGS standard sistemul de referință de observare UVN este sistem drept, iar volumul de vizualizare şi direcția de observare sunt orientate către axa – N.

O primă transformare geometrică este o schimbare de sistem de referinţă din sistemul de referinţă de observare UVN într-un sistem de referinţă notat Oxyz, cu centrul în centrul de proiecţie PRP şi cu axele paralele cu axele sistemului UVN (s-a ales această denumire pentru simplificarea notaţiilor, dar acest sistem nu este sistemul de referinţă universal pentru care, în mod obişnuit, s-a folosit notaţia Oxyz). Această transformare se efectuează printr-o translaţie inversă celei definite de vectorul de poziţie al punctului PRP.

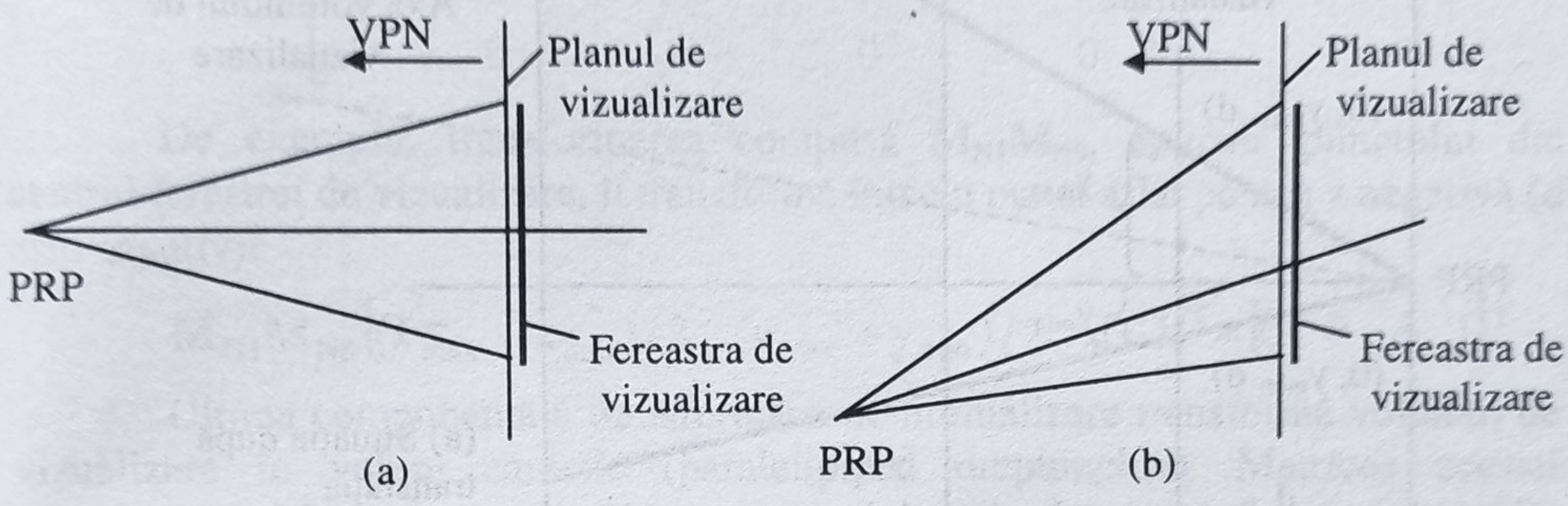


Fig. 4.9 (a) Proiecţia normală; (b) Proiecţie oblică.

Ca urmare a acestei transformări se modifică corespunzător valorile VPD, near, far, u_{min}, u_{max}, v_{min}, v_{max}. Fie d, n, f, x_{min}, x_{max}, y_{min}, respectiv y_{max}, valorile rezultate după translaţie. Aceste valori sunt numere reale şi pot fi pozitive sau negative. Matricea de transformare de normalizare pornind din acest sistem Oxyz se poate descompune în produs de trei matrice:

$$\mathbf{M_N = M_{N2}\, M_{N1} M_{N0}}$$

Matricele $\mathbf{M_{N1}}$ şi $\mathbf{M_{N2}}$ au semnificaţii asemănătoare cu matricele dezvoltate pentru sistemul de vizualizare precedent. Cealaltă transformare componentă, $\mathbf{M_{N0}}$, este o transformare de forfecare, prin care linia centrală a volumului de vizualizare (cea care uneşte centrul de proiecţie PRP cu centrul ferestrei de vizualizare, fig. 4.10) se suprapune cu axa z a sistemului.

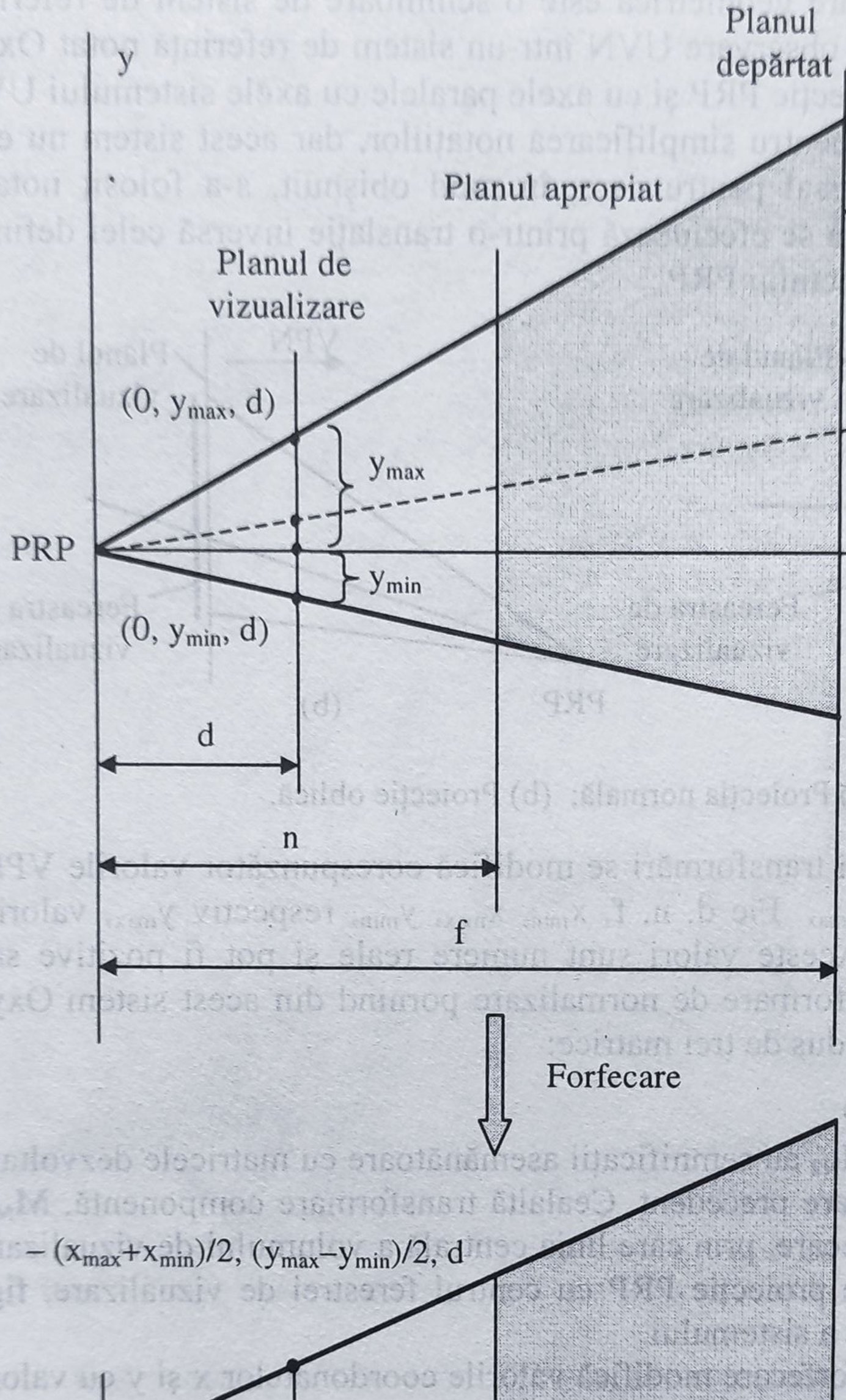
Transformarea de forfecare modifică valorile coordonatelor x şi y cu valori proporţionale cu z. Este simplu de observat că matricea:

$$\mathbf{M_{N0}} = \begin{bmatrix} 1 & 0 & -(x_{max} + x_{min})/2d & 0 \\ 0 & 1 & -(y_{max} + y_{min})/2d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.14)$$

efectuează o simetrizare a volumului de vizualizare. De exemplu, intersecţiile muchiilor orizontale ale ferestrei de vizualizare cu planul x = 0 se transformă astfel:

$$\mathbf{M_{N0}}[0 \quad y_{max} \quad d \quad 1]^T = [-(x_{max} + x_{min})/2 \quad (y_{max} - y_{min})/2 \quad d \quad 1]^T$$

$$\mathbf{M_{N0}}[0 \quad y_{min} \quad d \quad 1]^T = [-(x_{max} + x_{min})/2 \quad -(y_{max} - y_{min})/2 \quad d \quad 1]^T$$



(a) Situația după translația punctului PRP în origine.

(b) Situația după transformarea de forfecare. Volumul de vizualizare este simetric şi axa acestuia suprapusă peste axa z.

Fig. 4.10 Transformarea de forfecare în sistemul de vizualizare standard.

A doua transformare componentă este transformarea de scalare $\mathbf{M}_{N1}$, prin care volumul de vizualizare este transformat într-un volum simetric, cu unghiul de la vârful piramidei de 90°:

$$\mathbf{M}_{N1} = \begin{bmatrix} 2d/(x_{max} - x_{min}) & 0 & 0 & 0 \\ 0 & 2d/(y_{max} - y_{min}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

De exemplu, transformarea compusă $\mathbf{M}_{N1}\mathbf{M}_{N0}$, aplicată punctului din centrul ferestrei de vizualizare, îl transformă într-un punct aflat pe axa z negativă (d este negativ):

$$\mathbf{M}_{N1}\mathbf{M}_{N0}\left[(x_{max} + x_{min})/2 \quad (y_{max} + y_{min})/2 \quad d \quad 1\right]^T = \left[0 \quad 0 \quad d \quad 1\right]^T$$

Ultima componentă a transformării de normalizare transformă volumul de vizualizare în volum canonic (paralelipiped dreptunghic). Matricea acestei transformări, $\mathbf{M}_{N2}$, este asemănătoare matricei descrise în paragraful precedent, dar se ține seama de faptul că planul de vizibilitate apropiat nu mai coincide cu planul de vizualizare:

$$\mathbf{M}_{N2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-n) & -nf/(f-n) \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.16}$$

Sistemul normalizat rezultat este un sistem de referință stâng. În acest sistem se efectuează operațiile de decupare a obiectelor, după care secvența transformărilor necesare pentru generarea imaginii continuă cu transformarea în sistemul de referință ecran 3D.

4.4 SISTEMUL DE REFERINȚĂ ECRAN 3D

Transformarea în sistemul de referinţă ecran 3D efectuează o corespondenţă între fereastra din planul de vizualizare şi zona de afişare alocată pe display numită poartă de afişare (*viewport*). Fereastra de vizualizare este definită într-un sistem de referinţă bidimensional în planul $z_N = 0$ al sistemului de referinţă normalizat. Poarta de afişare este definită într-un sistem de referinţă bidimensional în planul $z_S = 0$ al sistemului de referinţă ecran 3D şi este, în general, o zonă rectangulară de dimensiuni egale cu numărul de pixeli corespunzători pe orizontală şi pe verticală a zonei afişate (fig. 4.11).

Transformarea din sistemul de referinţă normalizat în sistemul ecran 3D lasă coordonata z nemodificată. Sistemul de referinţă ecran 3D astfel definit este un sistem de referinţă stâng, la fel ca şi sistemul de referinţă normalizat.

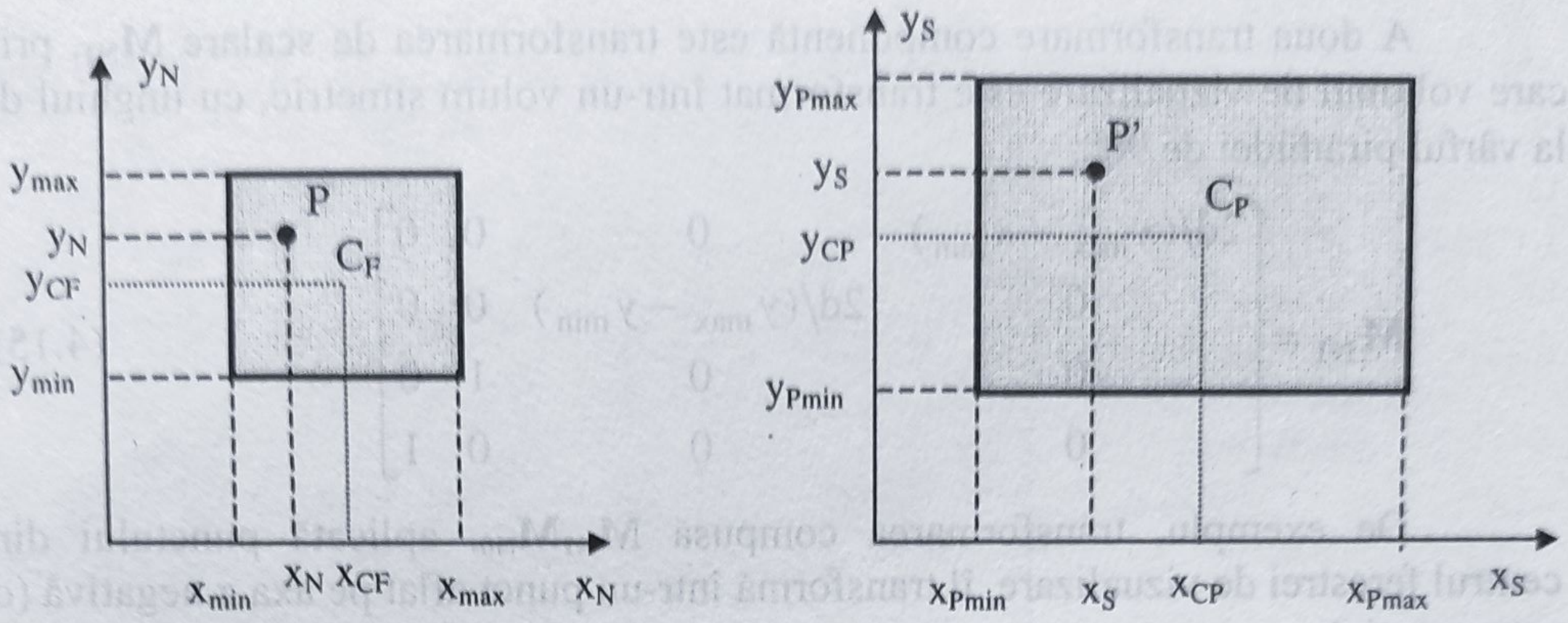


Fig. 4.11 Transformarea de la fereastra de vizualizare la poarta de afişare.

Pentru definirea transformării în sistemul de referinţă ecran 3D, se calculează mai întâi coordonatele centrului C_F al ferestrei şi ale centrului C_P al porţii:

$$x_{CF} = (x_{max} + x_{min})/2;\ y_{CF} = (y_{max} + y_{min})/2$$
$$x_{CP} = (x_{Pmax} + x_{Pmin})/2;\ y_{CP} = (y_{Pmax} + y_{Pmin})/2 \qquad (4.17)$$

Corespondenţa dintre coordonatele unui punct $P(x_N, y_N)$ din fereastră cu ale punctului $P'(x_S, y_S)$ în poarta de afişare se formulează matematic astfel:

$$\frac{x_S - x_{CP}}{x_N - x_{CF}} = \frac{x_{Pmax} - x_{Pmin}}{x_{max} - x_{min}};\ \frac{y_S - y_{CP}}{y_S - y_{CF}} = \frac{y_{Pmax} - y_{Pmin}}{y_{max} - y_{min}}$$

Se definesc factorii de scalare ai transformării:

$$s_x = \frac{x_{Pmax} - x_{Pmin}}{x_{max} - x_{min}}$$
$$s_y = \frac{y_{Pmax} - y_{Pmin}}{y_{max} - y_{min}} \qquad (4.18)$$
$$s_z = 1$$

Rezultă:

$$x_S = x_{CP} + s_x(x_N - x_{CF}) = s_x x_N + x_{CP} - s_x x_{CF}$$
$$y_S = y_{CP} + s_y(y - y_{CF}) = s_y y_N + y_{CP} - s_y y_{CF} \qquad (4.19)$$
$$z_S = z_N$$

Se poate deduce matricea de transformare ecran 3D prin compunerea a trei transformări succesive. Mai întâi se execută o translaţie, astfel încât centrul C_F al ferestrei să ajungă în origine, deci cu matricea de translaţie $\mathbf{T}(-x_{CF}, -y_{CF}, 0)$.

Fereastra centrată în origine se scalează cu factorii de scară s_x, s_y, definiţi de relaţia (4.18), astfel încât fereastra să ajungă la dimensiunea porţii, iar

coordonata z se lasă nemodificată. Matricea de scalare este deci $\mathbf{S}(s_x, s_y, s_z)$. Ultima transformare necesară este o translaţie, prin care centrul porţii este adus în punctul C_P, deci cu matricea de translaţie $\mathbf{T}(x_{CP}, y_{CP}, 0)$. Rezultă matricea de transformare din sistemul de referinţă normalizat în sistemul de referinţă ecran 3D:

$$\mathbf{M}_{NS} = \begin{bmatrix} 1 & 0 & 0 & x_{CP} \\ 0 & 1 & 0 & y_{CP} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_{CF} \\ 0 & 1 & 0 & -y_{CF} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{NS} = \begin{bmatrix} s_x & 0 & 0 & x_{CP} - s_x x_{CF} \\ 0 & s_y & 0 & y_{CP} - s_y y_{CF} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ w_S \end{bmatrix} = \mathbf{M}_{NS} \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix} \tag{4.20}$$

Din aceste relaţii rezultă $Z_S = Z_N$ şi $w_S = w_N$. În sistemul de referinţă ecran 3D se trece de la coordonatele omogene X_S, Y_S, Z_S, w_S, la coordonatele tridimensionale x_S, y_S, z_S prin împărţirea cu w_S.

Planul $z_S = 0$ din sistemul de referinţă ecran 3D este planul de proiecţie. Coordonatele x_S şi y_S reprezintă coordonatele proiecţiei ortografice în planul $z_S = 0$ (care este planul de porţii de afişare). Transformarea de proiecţie perspectivă din sistemul de referinţă de observare a devenit o transformare de proiecţie ortografică în sistemul de referinţă ecran 3D. Coordonata z_S în sistemul de referinţă ecran 3D este utilizată în algoritmii de eliminare a suprafeţelor ascunse. Volumul de vizualizare în sistemul ecran 3D este un paralelipiped dreptunghic cu baza un dreptunghi de dimensiuni egale cu dimensiunile porţii afişate şi înălţimea egală cu 1 ($z_S \in [0,1]$) (fig. 4.12).

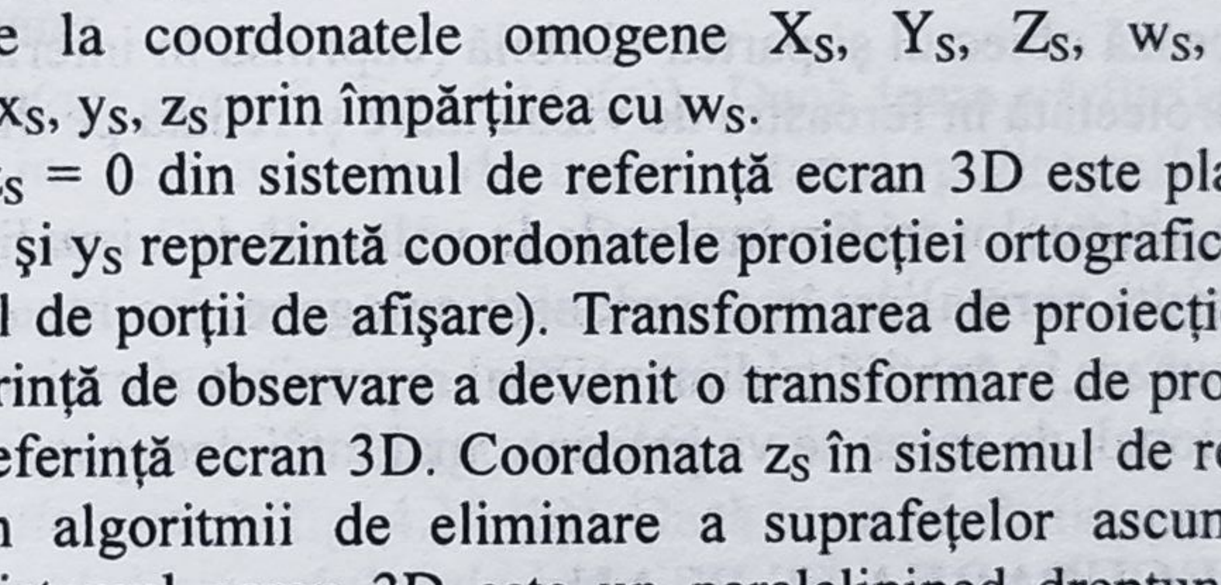


Fig. 4.12 Volumul de vizualizare în sistemul de referinţă ecran 3D.

4.5 DECUPAREA OBIECTELOR

Într-un sistem în care este definit punctul de observare şi un volum de vizualizare, nu interesează decât obiectele din interiorul acestui volum, a căror proiecţie se generează în fereastra de vizualizare. De aceea, în vizualizarea scenelor tridimensionale se execută operaţia de decupare la volumul de vizualizare (*clipping*), prin care se reţin numai obiectele sau părţile din obiecte cuprinse în acest volum.

Operaţia de decupare la volumul de vizualizare poate fi definită astfel: fiind dat un obiect tridimensional şi un volum de vizualizare, pot exista trei posibilităţi:

- Obiectul este complet în interiorul volumului şi, în acest caz, el este proiectat în fereastră şi redat pe display.
- Obiectul este complet în afara volumului de vizualizare şi, în acest caz, el poate fi ignorat.
- Obiectul intersectează volumul de vizualizare; în acest caz se decupează obiectul şi partea vizibilă (cuprinsă în interiorul volumului) este proiectată în fereastra de vizualizare şi redată pe display.

Decuparea obiectelor tridimensionale la volumul de vizualizare se execută în sistemul de referinţă normalizat în coordonate omogene, înainte de împărţirea cu w. Operaţia de decupare în spaţiul tridimensional reprezintă o extindere a decupării în planul bidimensional, de aceea se va prezenta mai întâi decuparea în plan.

4.5.1 DECUPAREA ÎN PLAN

Fiind dat un poligon oarecare în plan şi o regiune de decupare, problema decupării poligonului este de a reţine numai acea parte din poligon care se află în interiorul regiunii de decupare. Se consideră cazul cel mai frecvent întâlnit în care regiunea de decupare este un dreptunghi cu laturile paralele cu axele de coordonate. Această regiune de decupare corespunde ferestrei de vizualizare.

Majoritatea algoritmilor de decupare, atât în plan cât şi în spaţiu, se bazează pe algoritmul de decupare Sutherland-Hodgman [Suth74]. Algoritmul Sutherland-Hodgman împarte planul bidimensional în nouă regiuni, după poziţia faţă de regiunea dreptunghiulară de decupare dată prin ecuaţiile a patru drepte: $x = x_{min}$; $x = x_{max}$; $y = y_{min}$; $y = y_{max}$ (fig. 4.13).

Fiecărei regiuni îi corespunde un cod de vizibilitate (cunoscut sub numele de cod Sutherland-Cohen) reprezentat pe 4 biţi, b_0, b_1, b_2, b_3, definiţi astfel:

$$b_0 = semn(x - x_{min})$$
$$b_1 = semn(x_{max} - x)$$
$$b_2 = semn(y - y_{min})$$
$$b_3 = semn(y_{max} - y)$$

În continuare, codurile de vizibilitate se reprezintă printr-o secvenţă de cifre binare.

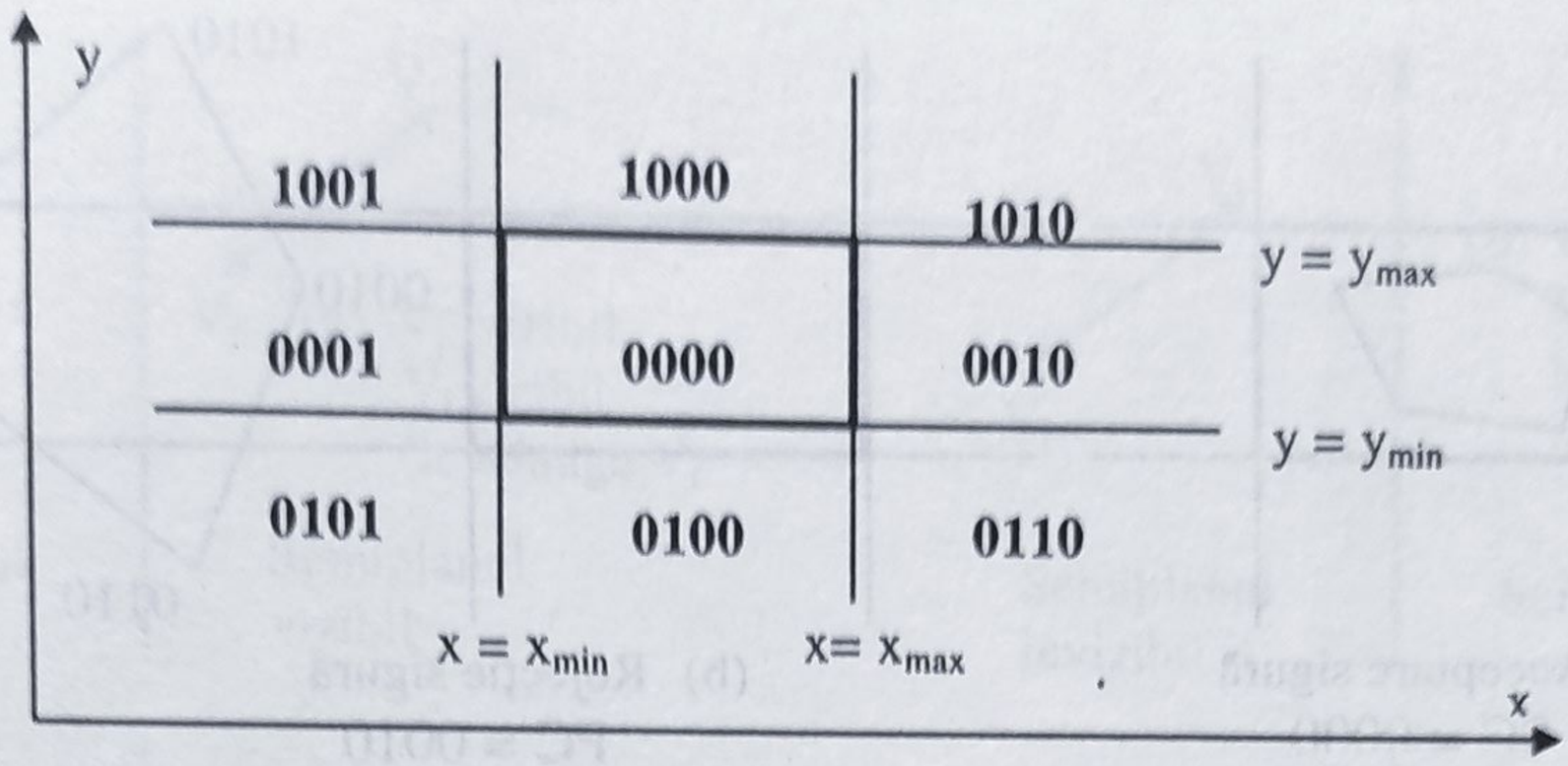


Fig. 4.13 Codurile de vizibilitate Sutherland-Cohen.

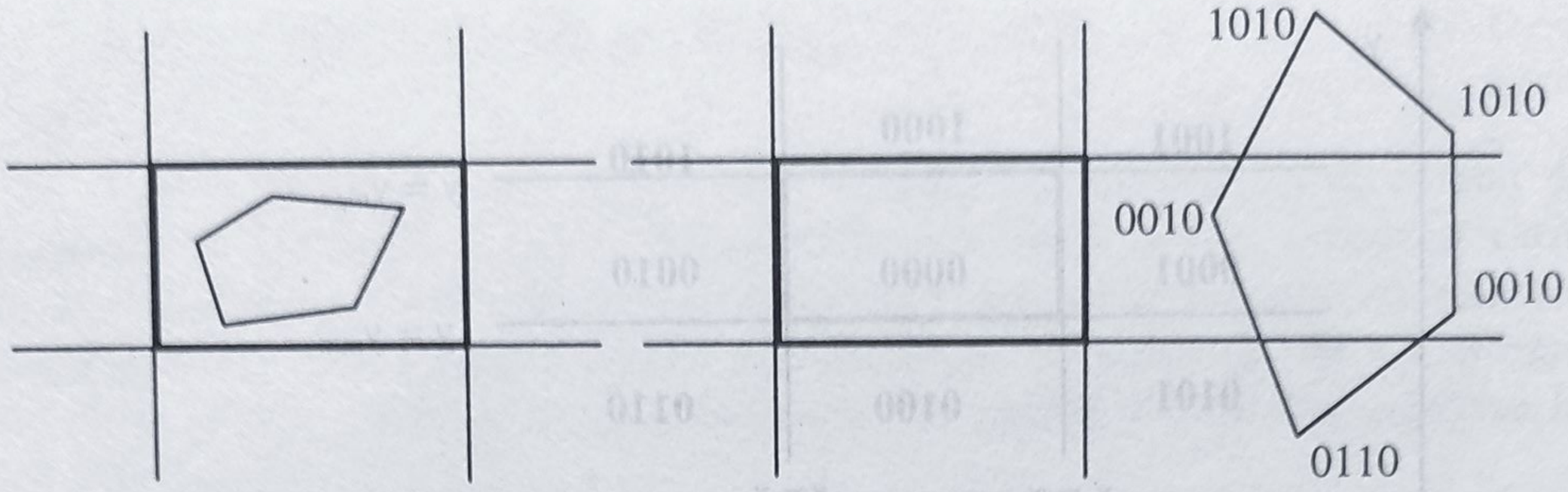
Un poligon oarecare din plan poate fi complet inclus în regiunea de decupare, poate fi complet în afara regiunii de decupare sau o poate intersecta. Folosind codurile de vizibilitate ale vârfurilor poligonului se pot identifica direct (fără să se execute calcule de intersecții) două situații ale poligonului față de regiunea de decupare:

- *Acceptare sigură* (fig. 4.14 (a)). Dacă toate vârfurile poligonului se află în regiunea de decupare, atunci polinomul este vizibil în întregime. Fie C1, C2,…., Cn, codurile de vizibilitate ale vârfurilor poligonului Pentru testul de acceptare sigură se calculează reuniunea acestor coduri (operația OR): SC = C1| C2 |…| Cn. Dacă SC = 0, atunci poligonul este sigur vizibil.
- *Rejecție sigură* (fig. 4.14 (b)). Dacă toate vârfurile unui poligon se află într-unul din semiplanele invizibile ($x < x_{min}$; $x > x_{max}$; $y < y_{min}$; $y > y_{max}$), atunci tot poligonul este sigur invizibil. Pentru testul de rejecție sigură se calculează intersecția codurilor de vizibilitate (operația AND): PC = C1 & C2 &…& Cn. Dacă PC ≠ 0, atunci poligonul este sigur invizibil deoarece toate punctele se află în semiplanul invizibil corespunzător bitului diferit de 0 din PC.

Dacă nu este îndeplinită nici condiția de acceptare sigură, nici condiția de rejecție sigură, atunci decuparea se calculează prin intersecția laturilor poligonului cu dreptele care mărginesc regiunea de decupare (fig. 4.14 (c)).

Dreapta (infinită) corespunzătoare unei laturi a regiunii de decupare împarte planul în două semiplane: semiplanul (regiunea) vizibil, care se află de aceeaşi parte a dreptei ca şi regiunea de decupare, şi semiplanul invizibil, care se află de partea opusă față de regiunea de decupare. De exemplu, pentru latura $x = x_{min}$, semiplanul $x < x_{min}$, este invizibil, iar semiplanul $x \geq x_{min}$ este vizibil.

În algoritmul de decupare Sutherland-Hodgman, decuparea unui poligon față de o regiune dată se efectuează într-un număr de paşi egal cu numărul de laturi ale regiunii de decupare. În fiecare pas se decupează poligonul (inițial sau provenit dintr-un pas anterior) cu o latură a regiunii de decupare: se elimină acea porțiune din poligon care se află în semiplanul invizibil corespunzător laturii şi se reține numai porțiunea vizibilă.



(a) Acceptare sigură
SC = 0000

(b) Rejecție sigură
PC = 0010

(c) Decupare prin intersecție
SC = 1111, PC = 0000

Fig. 4.14 Teste de vizibilitate pentru decuparea în plan.

Poligonul se reprezintă printr-o listă ordonată de vârfuri $V_1V_2...V_n$, şi fiecare pereche de două vârfuri consecutive (V_i V_{i+1}), pentru i < n, şi (V_nV_1) reprezintă o latură a poligonului ca un segment de dreaptă orientat (vector).

În fiecare pas de decupare se porneşte de la un poligon dat printr-o listă ordonată de vârfuri şi se creează lista vârfurilor poligonului rezultat prin decupare relativ la dreapta suport a unei laturi a regiunii de decupare. Inițial, lista vârfurilor poligonului rezultat este considerată vidă. Un segment orientat V_iV_j se poate afla într-una din patru situații posibile față de o latură de decupare (fig. 4.15).

(a) Dacă ambele vârfuri se află în semiplanul vizibil corespunzător laturii de decupare, atunci se adaugă în listă vârful V_j (fig. 4.15 (a)).

(b) Dacă ambele vârfuri se află în semiplanul invizibil, atunci nu se adaugă nici un vârf în listă (fig. 4.15 (b)).

(c) Dacă primul vârf al segmentului (V_i) se află în semiplanul vizibil, iar al doilea (V_j) se află în semiplanul invizibil (segmentul este orientat de la semiplanul vizibil către semiplanul invizibil), atunci se calculează intersecția I dintre segment şi dreapta de decupare şi se adaugă intersecția I în listă (fig. 4.15(c)).

(d) Dacă primul vârf al segmentului (V_i) se află în semiplanul invizibil, iar al doilea (V_j) se află în semiplanul vizibil (segmentul este orientat de la semiplanul invizibil către semiplanul vizibil), atunci se calculează intersecția I dintre segment şi dreapta de decupare. În lista vârfurilor se adaugă intersecția I şi apoi vârful V_j (fig. 4.15 (d)).

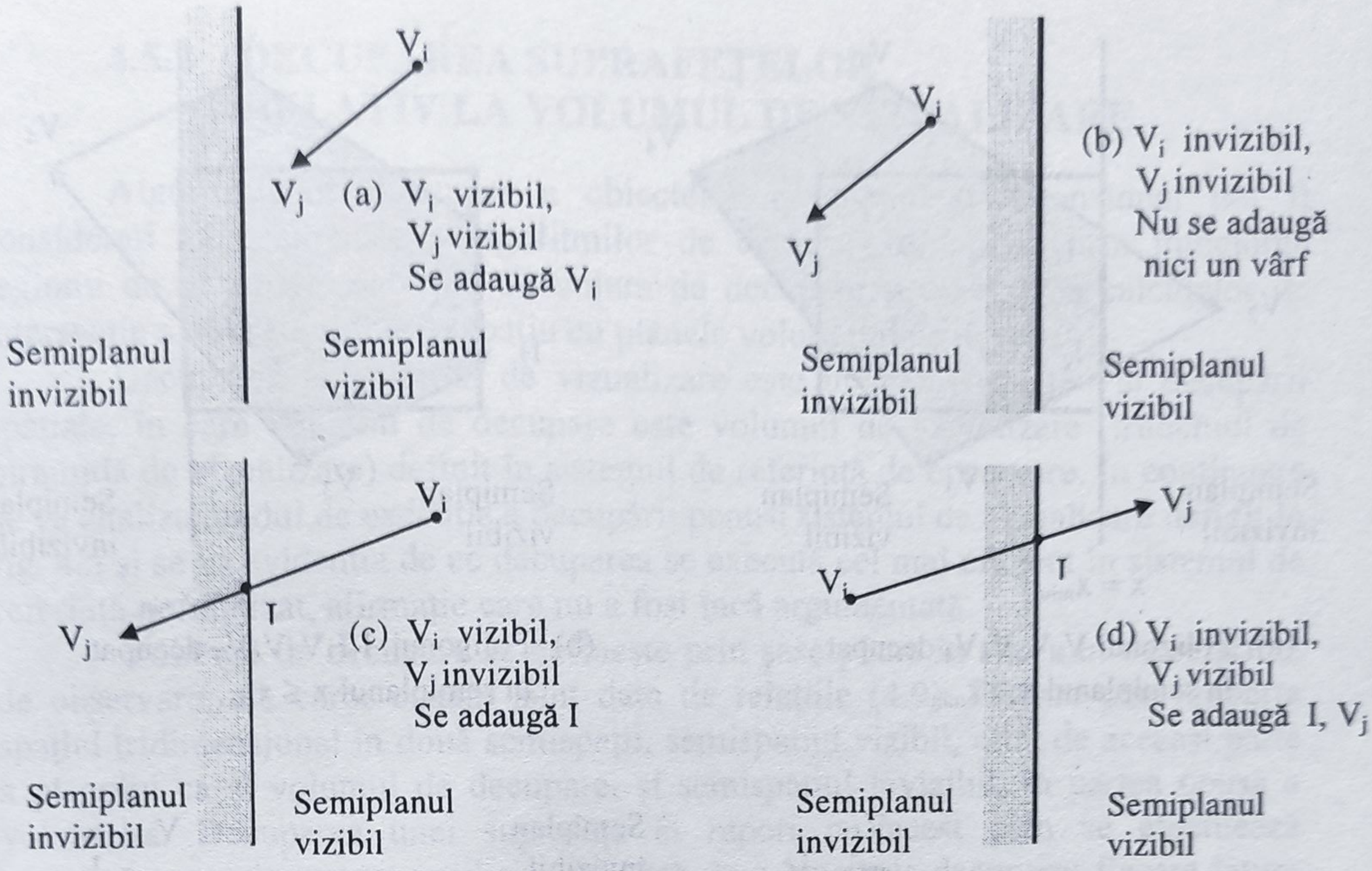


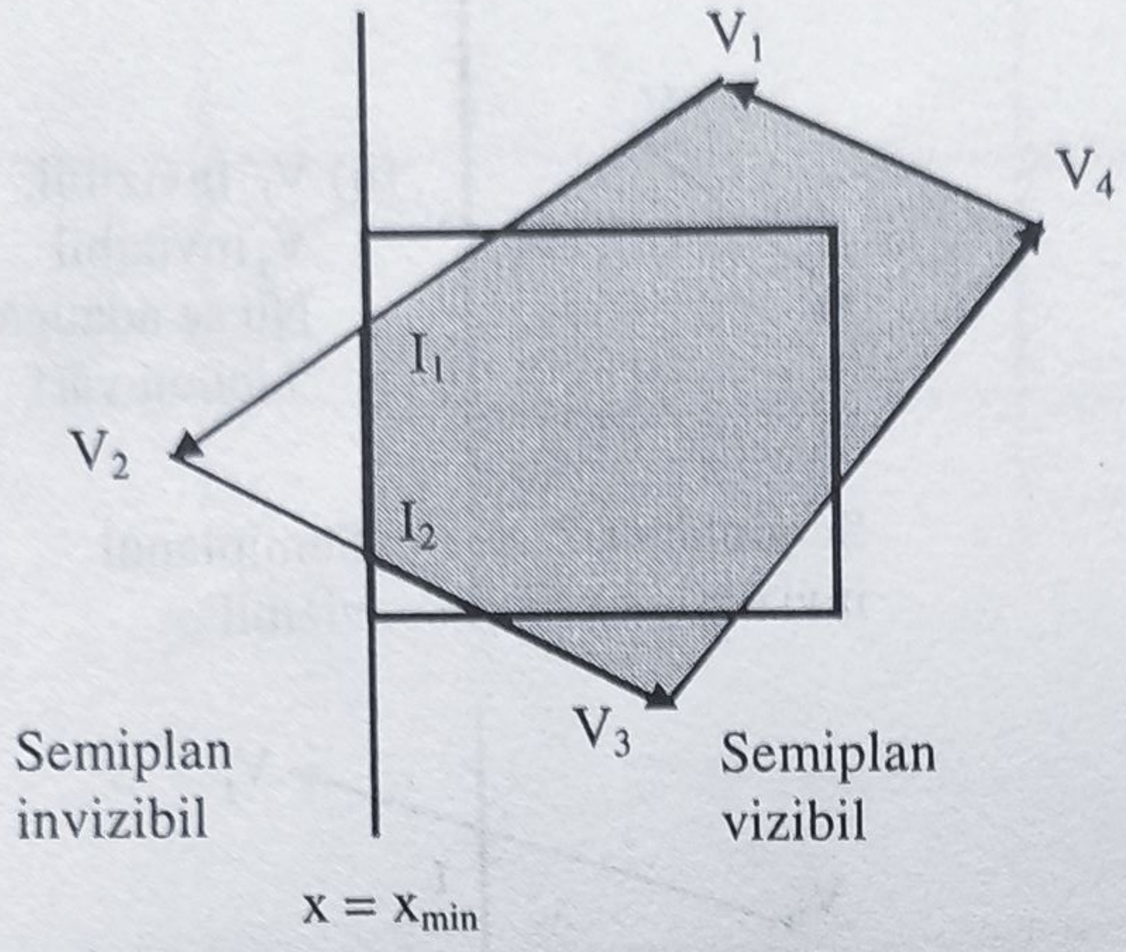
Fig. 4.15 Intersecția dintre o latură a poligonului și o dreaptă de decupare.

În primul pas, poligonul inițial $V_1V_2...V_n$ se decupează relativ la latura $x = x_{min}$ prin parcurgerea în ordine a tuturor segmentelor orientate ale poligonului (fig. 4.16(a)). Poligonul rezultat se decupează relativ la următoarea latură a regiunii de decupare (fig. 4.16 (b)). După parcurgerea unui număr de paşi egal cu numărul de laturi ale regiunii de decupare se obține poligonul complet decupat (fig. 4.16(d)).

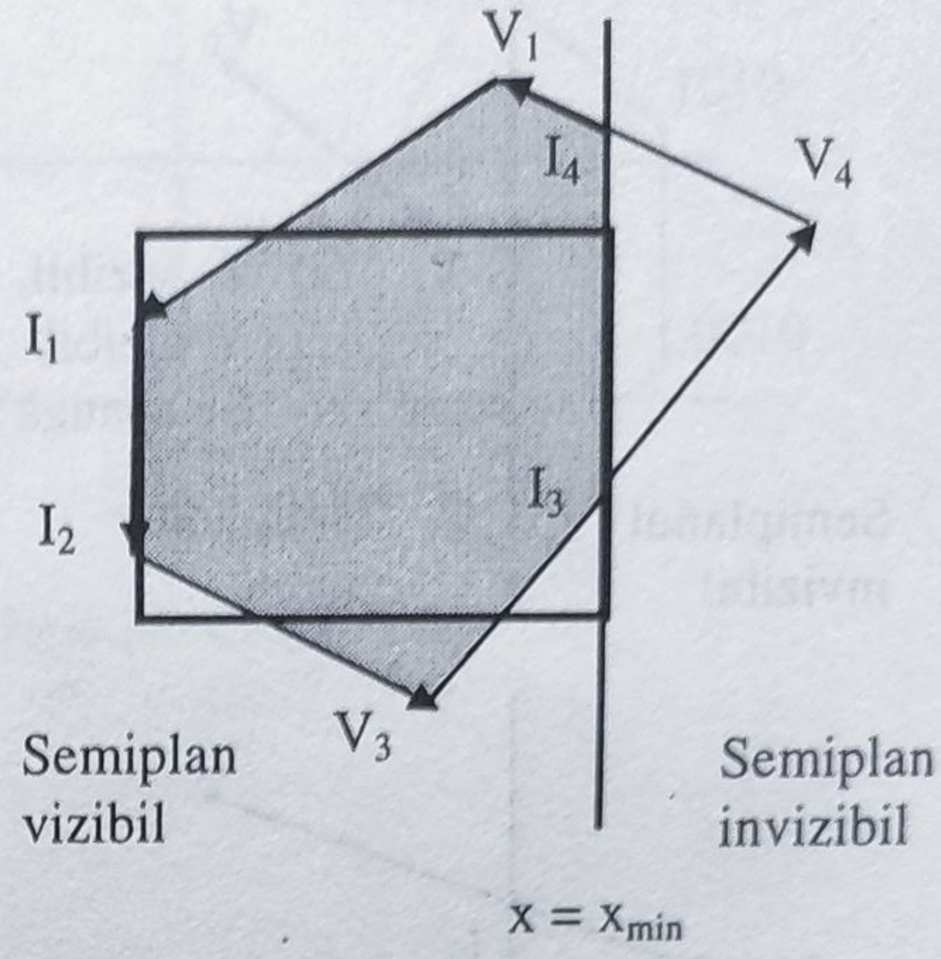
În fig.4.16 este ilustrat algoritmul de decupare Sutherland-Hodgman al unui poligon convex relativ la o regiune de decupare rectangulară, dar acest algoritm poate fi folosit în cazul general, al decupării oricărui tip de poligon (convex, concav sau cu găuri) relativ la o regiune de decupare oarecare.

Decuparea (în plan sau în spațiu) poate fi executată în orice operație care necesită construirea de poligoare sau obiecte tridimensionale noi prin calcul, pornind de la obiecte de bază, care sunt intersectate cu alte obiecte sau suprafețe.

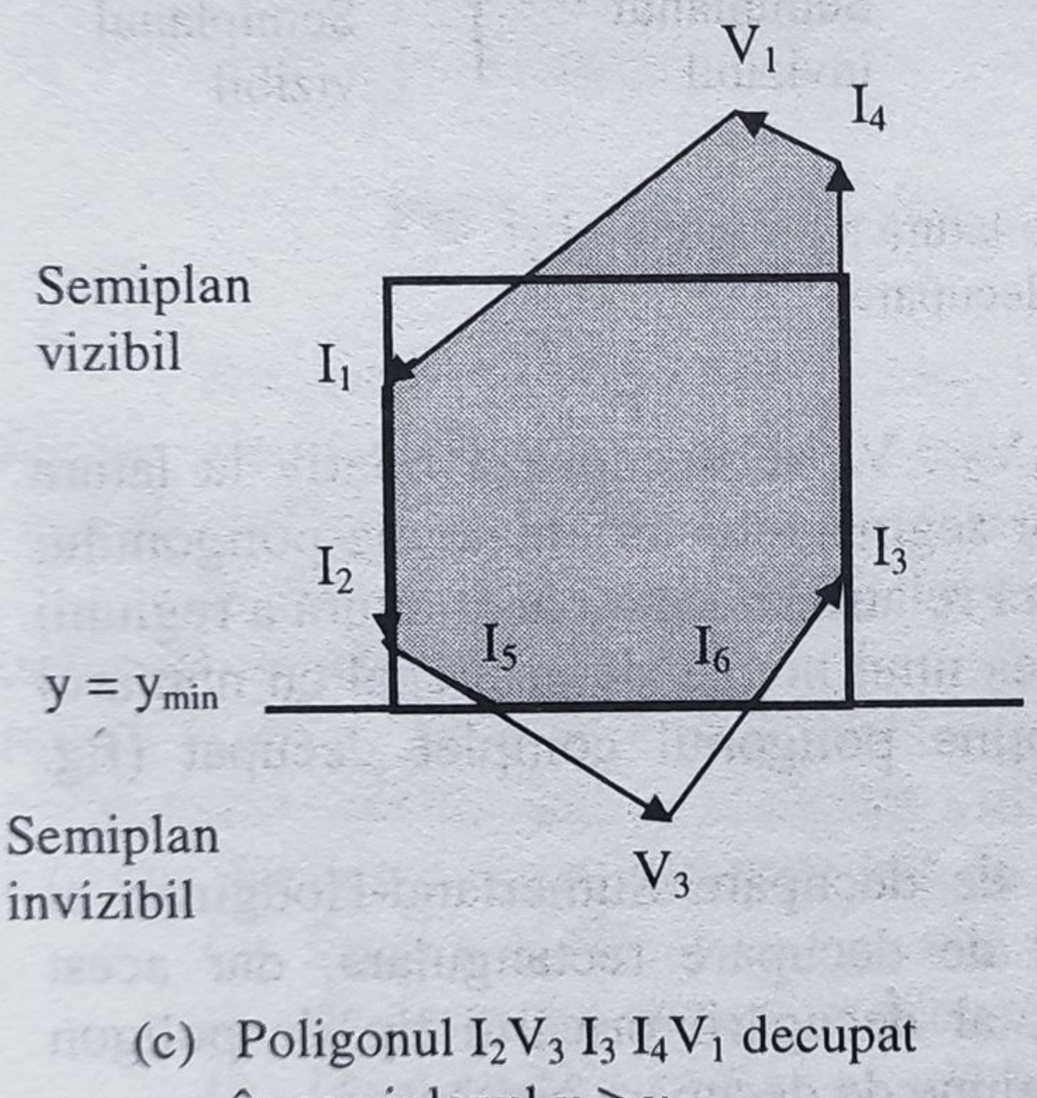
În generarea imaginilor bidimensionale, decuparea, ca operație componentă a secvenței de transformări de vizualizare, se efectuează relativ la fereastra de vizualizare, care este o regiune rectangulară. Poligoanele sau liniile rezultate, limitate la dimensiunea ferestrei, sunt transformate în sistemul de referință al porții de afişare şi redate pe display. Dacă nu se efectuează decuparea poligoanelor la limitele ferestrei, coordonatele vârfurilor în sistemul de referință al porții de afişare depăşesc limitele porții, iar rezultatul conversiei de rastru este incorect.



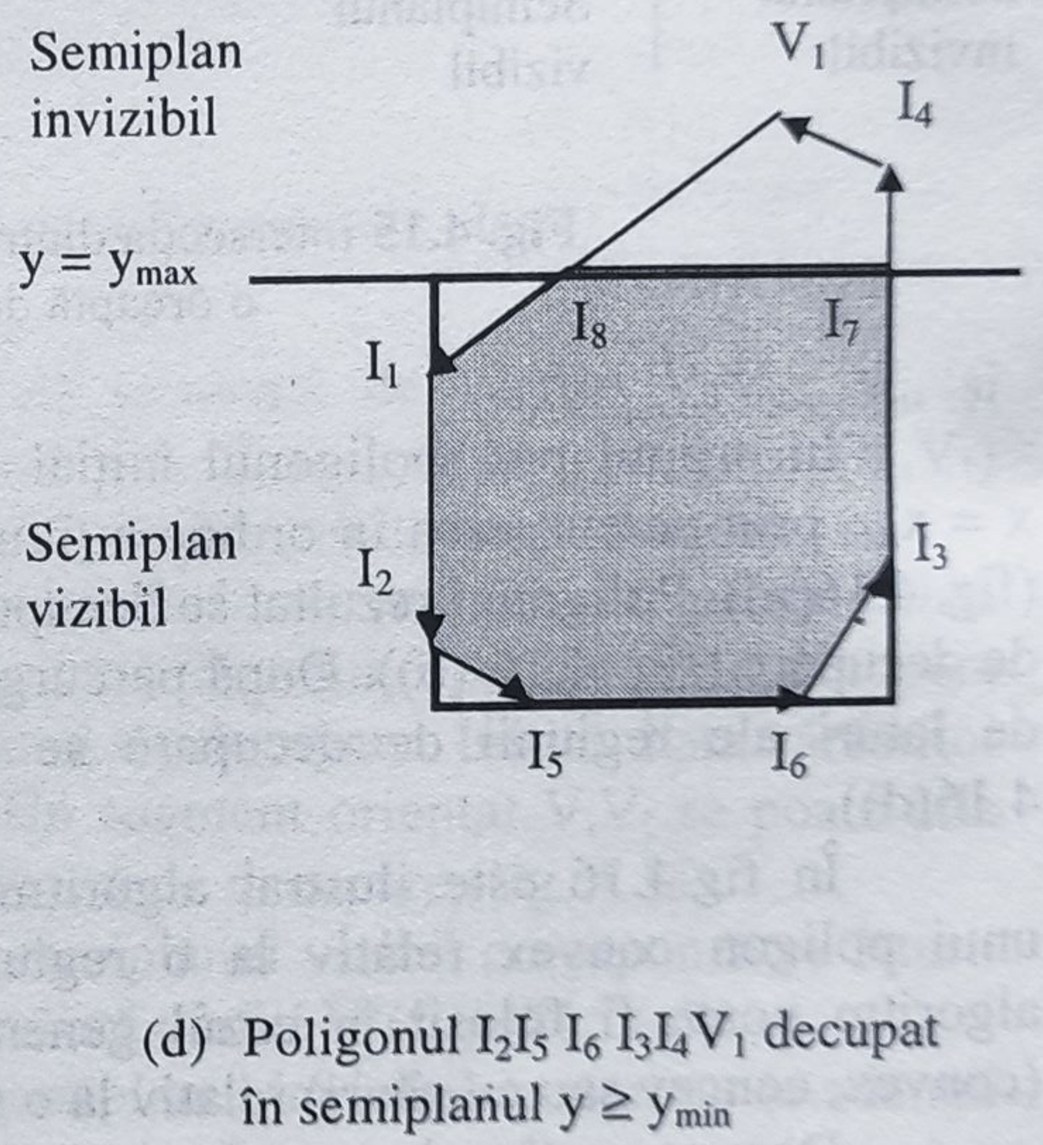
(a) Poligonul $V_1V_2V_3V_4$ decupat în semiplanul $x \geq x_{min}$



(b) Poligonul $I_1I_2V_3V_4V_1$ decupat în semiplanul $x \leq x_{max}$



(c) Poligonul $I_2V_3\ I_3\ I_4V_1$ decupat în semiplanul $y \geq y_{min}$



(d) Poligonul $I_2I_5\ I_6\ I_3I_4V_1$ decupat în semiplanul $y \geq y_{min}$

Fig. 4.16 Decuparea unui poligon în algoritmul Sutherland-Hodgman.
Poligonul rezultat este: $I_5\ I_6\ I_3I_7\ I_8\ I_1\ I_2$

Se poate efectua o decupare a poligoanelor şi liniilor şi în poarta de afişare, prin testul individual al coordonatelor fiecărui pixel şi rejectarea acelora care nu se încadrează în limitele porţii de afişare. Acesta este însă un procedeu deosebit de costisitor ca timp de execuţie şi nu se pune problema utilizării lui în grafica interactivă şi în realitatea virtuală.

4.5.2 DECUPAREA SUPRAFEȚELOR RELATIV LA VOLUMUL DE VIZUALIZARE

Algoritmii de decupare a obiectelor în spațiul tridimensional pot fi considerați ca o extensie a algoritmilor de decupare din plan, prin înlocuirea regiunii de decupare plane cu un volum de decupare şi efectuarea calculelor de intersecție a unei suprafețe în spațiu cu planele volumului de decupare.

Decuparea la volumul de vizualizare este un caz particular al decupării spațiale, în care volumul de decupare este volumul de vizualizare (trunchiul de piramidă de vizualizare) definit în sistemul de referință de observare. În continuare se va analiza modul de execuție a decupării pentru sistemul de vizualizare definit în fig. 4.5 şi se va evidenția de ce decuparea se execută cel mai eficient în sistemul de referință normalizat, afirmație care nu a fost încă argumentată.

Volumul de decupare se defineşte prin şase plane în sistemul de referință de observare, ale căror ecuații sunt date de relațiile (4.9). Fiecare plan împarte spațiul tridimensional în două semispații, semispațiul vizibil, aflat de aceeaşi parte a planului ca şi volumul de decupare, şi semispațiul invizibil, în partea opusă a volumului. Decuparea unei suprafețe în raport cu acest plan se efectuează asemănător cu decuparea unui poligon față de o latură de decupare: fiecare latură orientată introduce zero, unul sau două vârfuri în lista de vârfuri a suprafeței rezultate, în funcție de orientare şi de intersecția ei cu planul. Un vârf este vizibil în raport cu un plan dacă se află în semispațiul vizibil determinat de planul dat. De exemplu, un vârf este vizibil în raport cu planul $y_V = h\, z_V / d$, dacă $y_V \le h\, z_V / d$ (fig. 4.17 (a)). În mod asemănător, decuparea relativ la celelate plane ale volumului de decupare definit în sistemul de referință de observare stâng necesită testele:

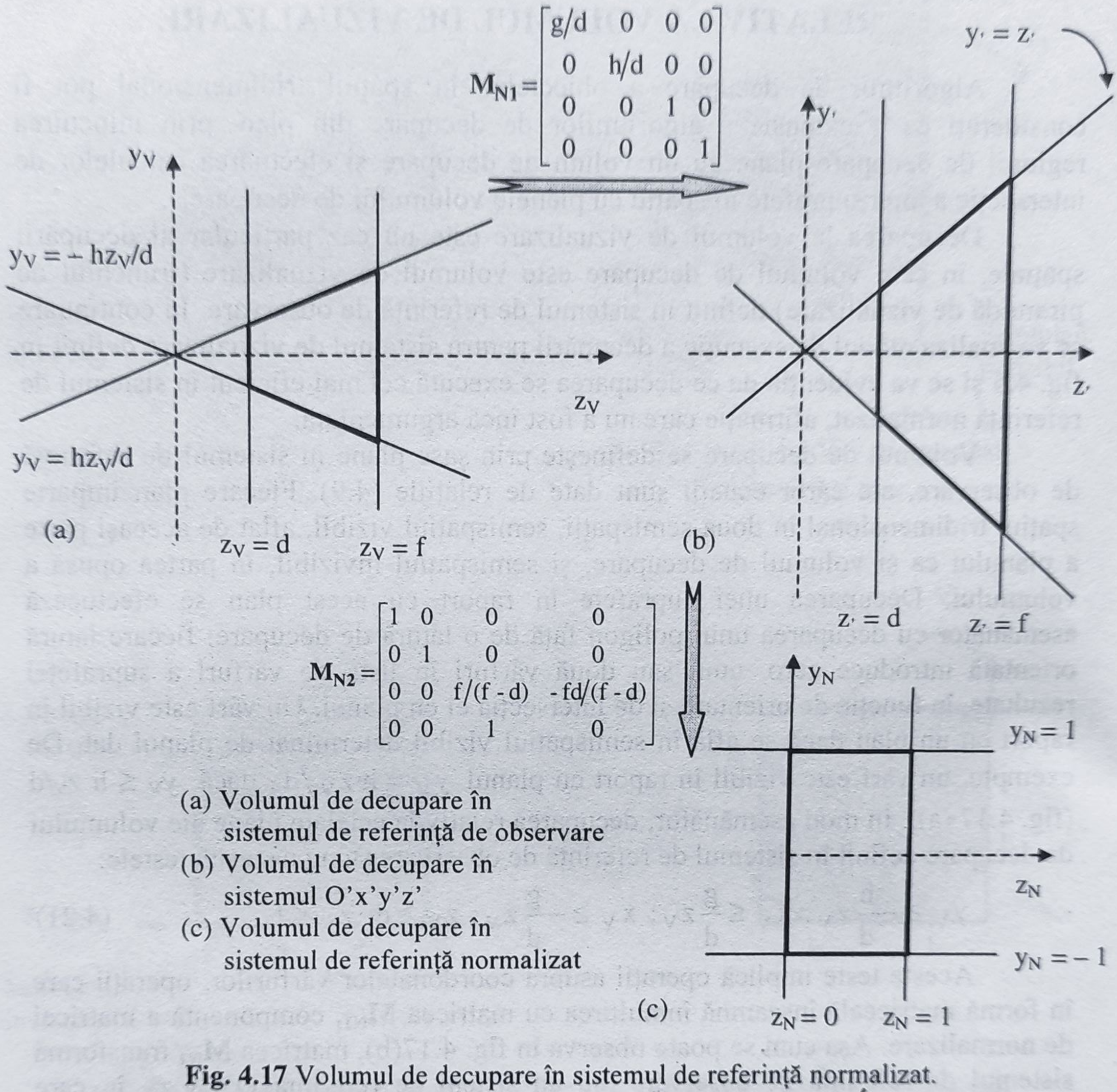
$$y_V \ge -\frac{h}{d} z_V;\ x_V \le \frac{g}{d} z_V;\ x_V \ge -\frac{g}{d} z_V;\ z_V \ge d;\ z_V \le f \qquad (4.21)$$

Aceste teste implică operații asupra coordonatelor vârfurilor, operații care în formă matriceală înseamnă înmulțirea cu matricea $\mathbf{M_{N1}}$, componentă a matricei de normalizare. Aşa cum se poate observa în fig. 4.17(b), matricea $\mathbf{M_{N1}}$ transformă sistemul de referință de observare într-un sistem de referință O'x'y'z', în care volumul de vizualizare (deci volumul de decupare) este un trunchi de piramidă regulată cu unghiul de la vârful piramidei de 90°. Relațiile (4.21) de testare a vizibilității unui punct în sistemul de referință de observare sunt echivalente cu următoarele relații în sistemul O'x'y'z':

$$x' \le z';\ x' \ge -z';\ y' \le z';\ y' \ge -z';\ z' \le f;\ z' \ge d \qquad (4.22)$$

Dacă s-ar executa decuparea în sistemul de referință intermediar O'x'y'z', atunci toate punctele din sistemul de referință de observare se transformă mai întâi cu matricea $\mathbf{M_{N1}}$, pentru trecerea în sistemul O'x'y'z', apoi se execută decuparea, iar după decupare, vârfurile vizibile se transformă cu matricea $\mathbf{M_{N2}}$, pentru trecere în sistemul de referință normalizat şi proiecție în fereastra de vizualizare (fig. 4.17(c)). Aplicarea separată a două transformări, mai întâi prin matricea $\mathbf{M_{N1}}$ şi apoi prin matricea $\mathbf{M_{N2}}$ este costisitoarte şi nenecesară, dat fiind că decuparea se poate face direct în sistemul normalizat (fig. 4.17 (c)).

$$M_{N1} = \begin{bmatrix} g/d & 0 & 0 & 0 \\ 0 & h/d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$M_{N2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-d) & -fd/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(a) Volumul de decupare în sistemul de referință de observare
(b) Volumul de decupare în sistemul O'x'y'z'
(c) Volumul de decupare în sistemul de referință normalizat

Fig. 4.17 Volumul de decupare în sistemul de referință normalizat. (Proiecție în planul x = 0).

În sistemul de referință normalizat volumul de decupare este un paralelipiped dreptunghic cu suprafețele paralele cu planele sistemului de coordonate. Testele de vizibilitate ale unui punct în raport cu planele volumului de decupare sunt:

$$x_N \le 1;\quad x_N \ge -1;\quad y_N \le 1;\quad y_N \ge -1;\quad z_N \le 1;\quad z_N \ge 0; \tag{4.23}$$

Aceste teste se pot efectua direct în coordonatele omogene în sistemul normalizat, fără să mai fie necesară împărțirea cu w_N:

$$\begin{aligned} &X_N \le w_N;\ X_N \ge -w_N; \\ &Y_N \le w_N;\ Y_N \ge -w_N; \\ &Z_N \le w_N;\ Z_N \ge 0; \end{aligned} \tag{4.24}$$

Codul de vizibilitate C al unui vârf în coordonate normalizate omogene se exprimă pe 6 biți şi fiecare bit se poziționează în funcție de rezultatul unuia din testele din relația (4.24):

$b_0 = semn(X_N - w_N)$
$b_1 = semn(w_N - X_N)$
$b_2 = semn(Y_N - w_N)$
$b_3 = semn(w_N - Y_N)$
$b_4 = semn(Z_N - w_N)$
$b_5 = semn(Z_N)$

Spațiul omogen normalizat se împarte în 27 de regiuni după poziția lor față de planele care definesc volumul canonic de vizualizare. În interiorul volumului canonic, toate punctele au codul de vizibilitate cu toți biții egali cu 0 (C = 000000). În toate celelalte regiuni, codul de vizibilitate este diferit de 0, având unul sau mai mulți biți egali cu 1.

Algoritmul Sutherland-Hodgman de decupare în spațiu se desfăşoară în mod asemănător cu algoritmul de decupare în plan. Pentru o suprafață dată prin vârfurile ei în spațiu, decuparea relativ la un volumul dat prin ecuațiile planelor sale se efectuează în modul următor:

(1) Se calculează codurile de vizibilitate ale vârfurilor suprafeței.
(2) Se efectuează testele de acceptare şi rejecție sigură. Pentru suprafețele care nu îndeplinesc nici unul din aceste teste se continuă cu decuparea prin intersecția cu volumul de decupare.
(3) Decuparea prin intersecție a unei suprafețe se efectuează într-un număr de paşi egal cu numărul de plane ale volumului de decupare. În fiecare pas se obține o nouă suprafață, din care a fost eliminată porțiunea invizibilă în raport cu planul de decupare respectiv.

După efectuarea decupării, se calculează coordonatele x_N, y_N, z_N ale vârfurilor suprafețelor vizibile rezultate (prin împărțirea cu w_N), iar toate celelalte vârfuri (ale suprafețelor invizibile) sunt ignorate. Acest lucru înseamnă că operația de împărțire omogenă se execută pentru un număr strict necesar de vârfuri, ceea ce reprezintă un câştig în viteza de execuție.

Se pune problema dacă nu ar fi mai simplu să fie efectuată decuparea în plan, deci după calculul proiecțiilor suprafețelor, dat fiind că algoritmul de decupare în plan este mai simplu. Acest lucru ar însemna ca toate suprafețele să fie proiectate pe planul de vizualizare, după care să se efectueze decuparea relativ la fereastra de vizualizare. Această soluție este însă inacceptabilă, datorită faptului că în fereastra de proiecție se proiectează atât punctele (vârfuri ale suprafețelor) aflate în volumul de vizualizare, cât şi cele aflate într-un volum simetric cu acesta față de planul $z_V = 0$, care se află în spatele observatorului.

Tratarea tuturor acestor suprafețe proiectate în fereastra de vizualizare dar invizibile consumă un timp de execuție în mod inutil. De aceea, decuparea obiectelor la volumul de vizualizare nu se poate executa corect şi eficient decât în spațiul tridimensional, şi anume în coordonate normalizate omogene.

4.5.3 ELIMINAREA OBIECTELOR AFLATE ÎN EXTERIORUL VOLUMULUI DE VIZUALIZARE

Se poate detalia modul de execuţie a decupării relativ la volumul de vizualizare. Algoritmul de bază se referă la decuparea unei suprafeţe în spaţiu, dată ca o succesiune de vârfuri şi, implicit, o succesiune de segmente de dreaptă orientate (laturile suprafeţei), relativ la un volum de decupare, care este volumul canonic în coordonate omogene normalizate.

Decuparea punctelor în spaţiu este un simplu test de vizibilitate (ecuaţiile 4.24), iar decuparea unei linii este un caz particular (cu n = 2) al algoritmului de decupare a unei suprafeţe.

Decuparea unui obiect tridimensional, modelat ca o colecţie de feţe (suprafeţe plane), se reduce la decuparea fiecărei feţe în parte, iar obiectul rezultat este compus din toate feţele sale vizibile şi părţile rezultate prin decupare.

O creştere a eficienţei în operaţia de decupare se poate obţine folosind teste de rejecţie şi de acceptare sigură nu numai pentru fiecare suprafaţă în parte, dar şi pentru un obiect tridimensional în întregime.

Procedeul de calcul al reuniunii (SC) şi al intersecţiei (PC) codurilor de vizibilitate ale unui obiect tridimensional depinde de modul de reprezentare a acestuia. Pentru obiectele tridimensionale modelate prin feţe date ca listă de indecşi la vârfuri se calculează direct valorile SC şi PC ale celor N vârfuri ale obiectului. Pentru obiecte reprezentate prin feţe separate, se pot colecta codurile de vizibilitate ale feţelor componente şi calcula reuniunea SC şi intersecţia PC. Funcţia de acceptare sau rejecţie sigură (Culling) a unui obiect tridimensional este prezentată în pseudocod asemănător limbajului C astfel:

```
int Culling(){
    PC = 0x3F;
    SC = 0;
    for (i=0; i<N; i++){
        SC = SC | Ci;
        PC = PC & Ci;
    }
    if (PC) {
        Obiect sigur invizibi, va fi rejectat;
        return 0;
    }
    else if (SC){
        Obiect sigur vizibil în întregime;
        Nu se va mai testa fiecare suprafaţă;
        return 1;
    }
    else {
        Se va decupa fiecare suprafaţă;
        return -1;
    }
}
```

Ce anume se execută în fiecare din situaţii depinde, evident, de modul de organizare al programului şi de biblioteca grafică folosită. În orice caz, dacă obiectul este sigur invizibil, atunci se abandonează toate operaţiile referitoare la acest obiect şi se trece la obiectul următor.

Selecţia obiectelor, adică identificarea obiectelor potenţial vizibile pentru redarea imaginii acestora şi eliminarea obiectelor sigur invizibile (*culling*), este o resursă puternică de creştere a eficienţei de redare a scenelor virtuale. În scenele virtuale de dimensiuni mari, care conţin un număr mare de obiecte, în fiecare cadru de imagine sunt vizibile şi deci trebuie să fie prelucrate un număr mult mai mic obiecte decât numărul total de obiecte ale scenei. De aceea, identificarea cât mai devreme posibil a obiectelor sigur invizibile şi eliminarea lor conduce la reducerea substanţială a timpului de redare a scenei. În redarea scenelor complexe, acest procedeu se implementează mai eficient decât modul simplu descris anterior prin definirea volumului de delimitare a obiectelor.

4.5.4 VOLUMUL DE DELIMITARE

Volumul de delimitare (*bounding box*) al unui obiect sau al unui grup de obiecte (ierarhie de obiecte) este un volum definit cât mai simplu, care include toate vârfurile obiectelor şi are dimensiuni minime. Se folosesc ca volume de delimitare paralelipipede dreptunghice sau sfere.

Fiind dat un obiect cu un volum de delimitare, testul de eliminare (*culling*) se efectuează asupra volumului de delimitare: se calculează intersecţia codurilor de vizibilitate ale vârfurilor volumului de delimitare şi, dacă este diferită de zero, atunci volumul de delimitare şi, implicit, întregul obiect este invizibil şi abandonat.

Această modalitate de reprezentare şi prelucrare a obiectelor este aproape unanim adoptată în realitatea virtuală, datorită execuţiei eficiente a testului: decizia de eliminare a unui obiect se poate lua prin considerarea unui număr de opt vârfuri, în locul unui număr foarte mare de vârfuri cât are un obiect în mod obişnuit.

Volumul de delimitare al unui obiect se defineşte în sistemul de referinţă local (de modelare) al obiectului, ca un paralelipiped dreptunghic cu muchiile paralele cu axele de coordonate. Un astfel de paralelipiped se poate specifica prin coordonatele a două vârfuri opuse (x_{min}, y_{min}, z_{min}) şi (x_{max}, y_{max}, z_{max}). Vârfurile lui sunt transformate din sistemul de referinţă local în alte sisteme de referinţă (universal, de observare, etc.), la fel ca şi vârfurile obiectului. Testul de eliminare se poate efectua în sistemul de referinţă de observare sau în alt sistem de referinţă (sistemul normalizat).

În fig. 4.18 sunt reprezentate trei obiecte în poziţii diferite faţă de volumul de vizualizare. Volumul de delimitare al primului obiect este complet exterior faţă de volumul de vizualizare şi obiectul corespunzător (ceainic) este ignorat pentru punctul de observare dat. Cel de-al doilea obiect (icosaedru) este complet vizibil, volumul său de delimitare fiind inclus în volumul de vizualizare. Pentru cel de-al treilea obiect (sferă), volumul de delimitare intersectează volumul de vizualizare şi trebuie să fie executată decuparea suprafeţelor sale relativ la volumul de vizualizare.

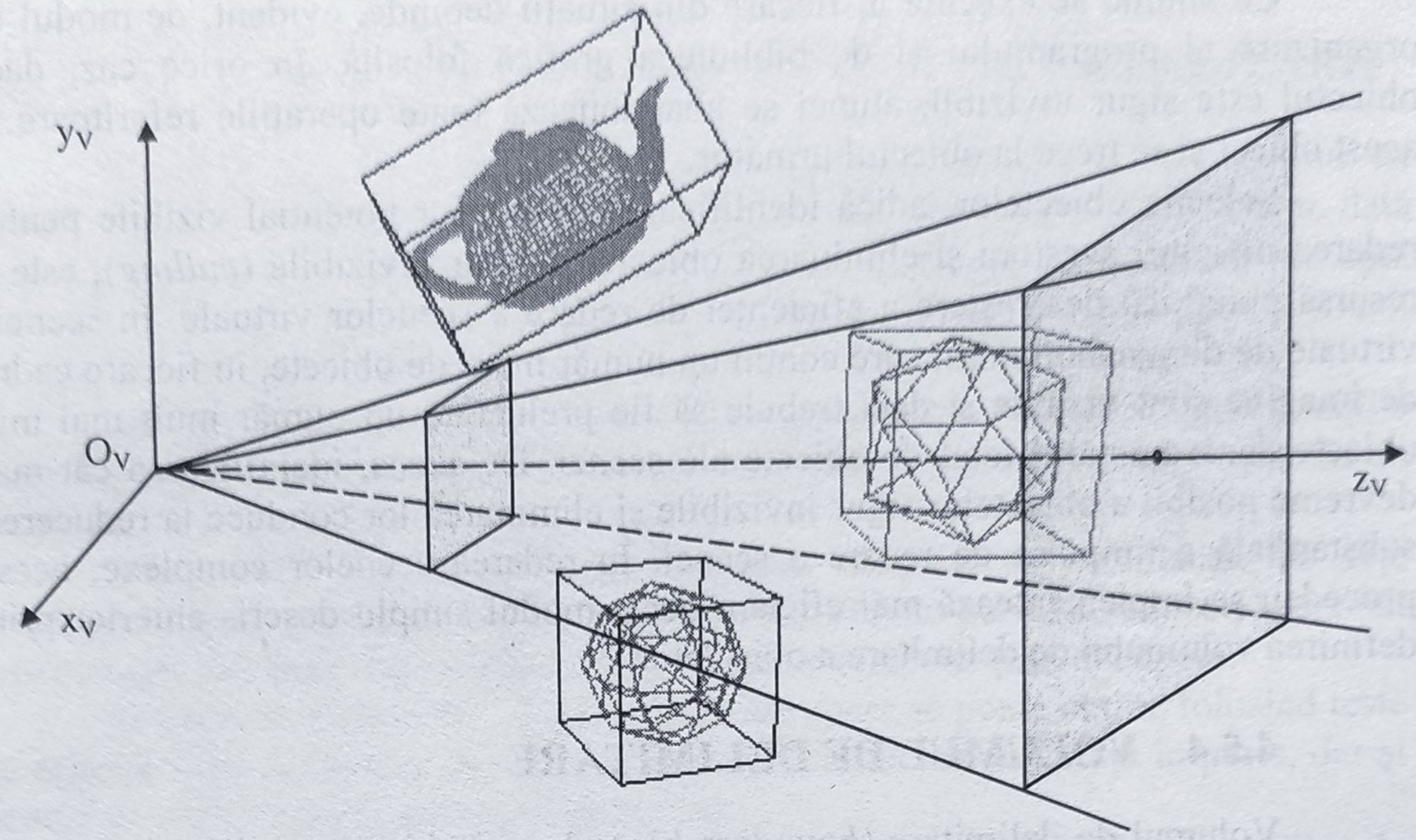


fig. 4.18 Eliminarea obiectelor pe baza volumului de delimitare (*bounding box*).

În scenele virtuale modelate ierarhic se construiesc volume de delimitare la fiecare nivel de ierarhie şi obiectele sau grupurile de obiecte sunt selectate pe baza testului de eliminare efectuat asupra volumului de delimitare corespunzător.

4.5.5 DETECȚIA COLIZIUNII

Detecția coliziunii este un aspect important în modelarea mediului virtual, prin faptul că simulează un aspect realist al mediului: obiectele reale nu pot trece unele prin altele; atunci când un obiect se atinge de un altul există un răspuns de coliziune: deformarea obiectelor, schimbarea traiectoriei de deplasare, explozia, etc. La interacțiunea dintre utilizator şi obiectele virtuale se pot produce forțe, vibrații şi alte reacții tactile care sunt prelucrate în sistem.

Detecția coliziunii se poate calcula aproximativ, folosind volumele de delimitare, sau exact, prin considerarea tuturor punctelor obiectelor. Dat fiind că volumul de delimitare depăşeste în multe puncte suprafața de frontieră a obiectului, calculul aproximativ al coliziunii introduce coliziuni false atunci când nu obiectele s-au atins, ci volumele lor de delimitare.

Detecția exactă a coliziunii obiectelor se efectuează după calculul coliziunii între volumele de delimitare. Dacă două obiecte au volumele de delimitare disjuncte, atunci ele sigur nu se ating. Dacă două obiecte au volume de delimitare care se ating sau interpătrund, atunci se calculează coliziunea exactă prin considerarea tuturor punctelor obiectelor.

TRANSFORMAREA DE RASTRU

Transformarea de rastru (*rasterization*) este o conversie de la reprezentarea prin coordonatele vârfurilor a unui segment sau a unui poligon (primitivă geometrică), la reprezentarea prin mulțimea corespunzătoare de pixeli care se afișează pe display. Transformarea de rastru se mai numeşte conversie de baleiere (*scan conversion*), sau desenarea segmentelor, respectiv a poligoanelor (*line drawing*, *polygon drawing*) sau generare linie, respectiv poligon (*line generation*, *polygon generation*).

Transformarea de rastru implică trecerea de la spațiul bidimensional continuu (poarta de afişare definită în planul $x_S = 0$ al sistemului de referință ecran 3D) la spațiul discret al imaginii (fig. 5.1)

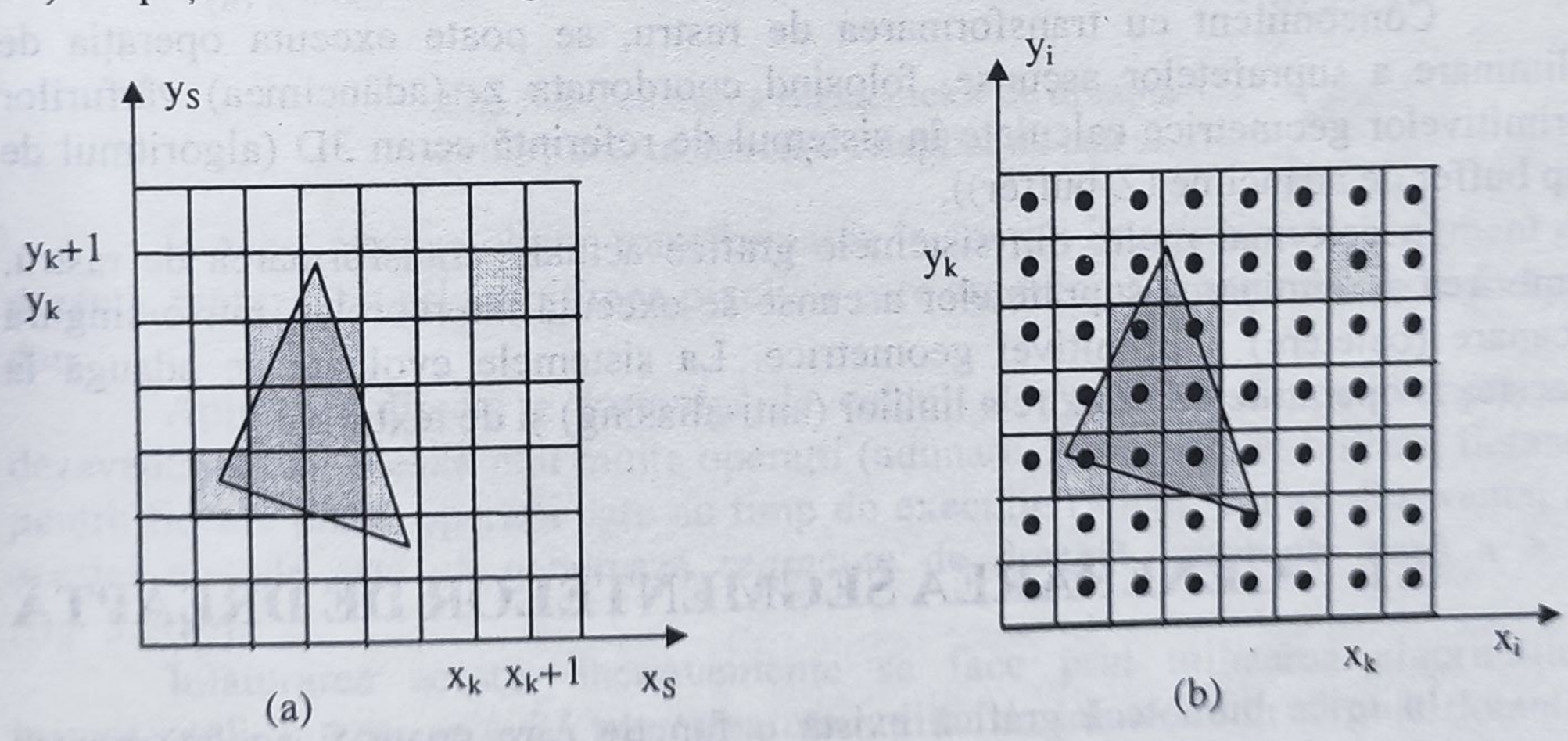


Fig. 5.1 (a) Spațiul bidimensional continuu al porții de afişare;
(b) Spațiul bidimensional discret al imaginii.

Spațiul imagine este un tablou bidimensional de locații discrete numite pixeli, fiecare pixel având o adresă care corespunde poziției în imagine a acestuia. Dimensiunea imaginii este specificată prin numărul de pixeli pe orizontală (M),

respectiv pe verticală (N), care pot fi afişaţi în zona de ecran atribuită porţii de afişare. Dimensiunea maximă a imaginii depinde de rezoluţia display-ului folosit.

Un pixel ocupă în poarta de afişare o zonă pătrată de dimensiune 1 × 1 a cărui adresă este dată de coordonata colţului stânga-jos. De exemplu, pixelul de adresă (x_k, y_k) din spaţiul imagine corespunde unui pătrat din poarta de afişare cu colţurile de coordonate (x_k, y_k) (x_k+1, y_k) (x_k+1, y_k+1) (x_k, y_k+1), unde x_k şi y_k sunt numere întregi. În mod simplificat se consideră că imaginea pe ecran a pixelilor este formată din zone pătrate adiacente, dar, în realitate, imaginea pe ecran a unui pixel este o zonă circulară, iar aceste cercuri nu sunt adiacente ci se întrepătrund.

Un pixel din spaţiul imagine se reprezintă prin culoarea lui, care se memorează într-un buffer de imagine. Adresa pixelului în spaţiul imagine determină adresa în bufferul de imagine în care se memorează culoarea acestuia.

Din coordonatele x şi y ale vârfurilor primitivelor geometrice în poarta de afişare se calculează ecuaţiile segmentelor de dreaptă (laturile), iar mulţimea de pixeli generată în spaţiul imagine trebuie să aproximeze cât mai bine segmentele de dreaptă sau interiorul poligoanele delimitat prin laturile sale (fig.5.1).

Transformarea de rastru a segmentelor sau poligoanelor calculează adresa fiecărui pixel din mulţimea de pixeli prin care se aproximează primitiva geometrică dată şi înscrie la adresa corespunzătoare din bufferul de imagine intensitatea de culoare a pixelului respectiv. Culoarea pixelului poate fi o culoare constantă, rezultată din atributul de culoare al primitivei geometrice sau din condiţii de umbrire constantă (poligonală), şi atunci această culoare se înscrie la toate adresele pixelilor care aparţin primitivei geometrice în bufferul de imagine. În cazul umbririi Gouraud, culoarea fiecărui pixel se calculează prin interpolare, concomitent cu calculul adresei pixelilor în transformarea de rastru.

Concomitent cu transformarea de rastru, se poate executa operaţia de eliminare a suprafeţelor ascunse, folosind coordonata z_S (adâncimea) vârfurilor primitivelor geometrice calculate în sistemul de referinţă ecran 3D (algoritmul de tip buffer de adâncime (Z-buffer)).

În cele mai multe din sistemele grafice actuale, transformarea de rastru, umbrirea şi eliminarea suprafeţelor ascunse se execută intercorelat, într-o singură scanare (baleiere) a primitivei geometrice. La sistemele evoluate se adaugă la acestea şi operaţiile de netezire a liniilor (anti-aliasing) şi de texturare.

5.1 GENERAREA SEGMENTELOR DE DREAPTĂ

În orice bibliotecă grafică există o funcţie care poate fi apelată pentru generarea unui segment de dreaptă. De exemplu funcţia line() din biblioteca compilatorului Borland C, sau funcţia LineTo() din interfaţa Win32API, trasează o linie în ferestra de afişare a programului.

Se consideră un segment de dreaptă AB dat prin capetele lui (x_A, y_A), (x_B, y_B) în sistemul de coordonate poarta de afişare (fig. 5.2). Ecuaţia dreptei care trece prin aceste puncte este $y = ax + b$, unde $a = (y_B - y_A) / (x_B - x_A)$, $b = y_A - a\, x_A$.

În același sistem de referință, adresa unui pixel este dată de coordonatele colțului lui stânga-jos, deci pixelul de adresă (x_k, y_k) este un pătrat cu colțurile (x_k, y_k), (x_k+1, y_k), (x_k+1, y_k+1), (x_k, y_k+1), unde x_k și y_k sunt numere întregi.

Dacă $x_1 \le x_2$, atunci se calculează secvența de puncte pe dreaptă:

$(x_1 = x_A, y_1 = ax_1 + b)$,

$(x_2 = x_1+1, y_2 = ax_2 + b), \ldots (x_i, y_i = ax_i + b) \ldots\ldots$

Fiecărui punct pe dreaptă i se asociază un pixel a cărui adresă în sistemul de referință imagine se aproximează prin trunchierea sau rotunjirea coordonatelor punctului în sistemul de referință de afișare:

$x_i' = (int)x_i$ sau $x_i' = (int)(x_i + 0.5)$

$y_i' = (int)y_i$ sau $y_i' = (int)(y_i + 0.5)$

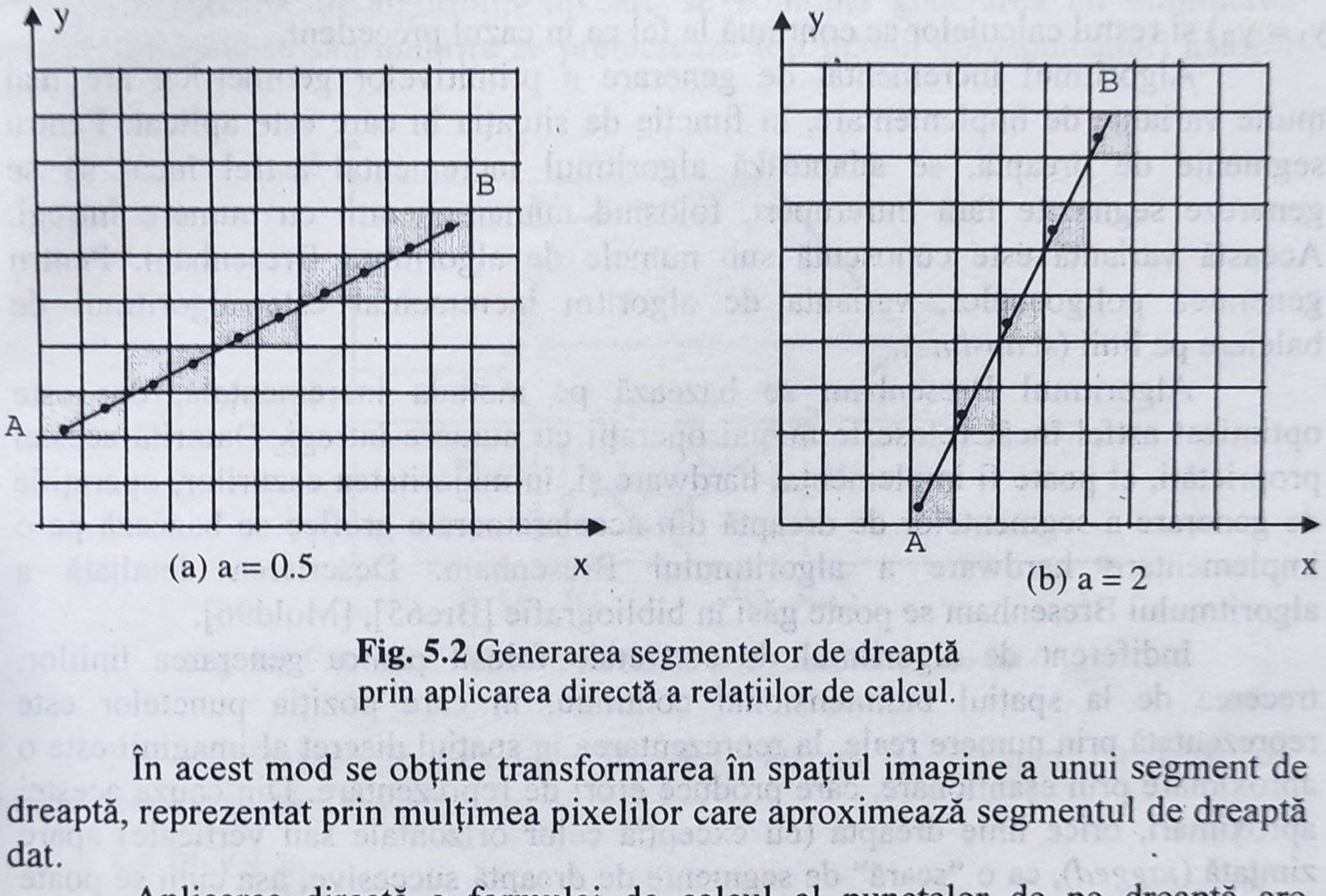


Fig. 5.2 Generarea segmentelor de dreaptă prin aplicarea directă a relațiilor de calcul.

În acest mod se obține transformarea în spațiul imagine a unui segment de dreaptă, reprezentat prin mulțimea pixelilor care aproximează segmentul de dreaptă dat.

Aplicarea directă a formulei de calcul al punctelor de pe dreaptă are dezavantajul că necesită mai multe operații (adunare, înmulțire) în numere flotante pentru fiecare pixel, operații care au timp de execuție ridicat. Un alt dezavantaj al acestei metode este că generează segmente de dreaptă întrerupte dacă $a > 1$ (fig. 5.2(b)).

Înlăturarea acestor inconveniente se face prin utilizarea algoritmilor incrementali, în care se evită repetarea operațiilor de înmulțire în virgulă flotantă, folosind trecerea prin incrementare de la un punct la următorul. Pentru obținerea segmentelor de dreaptă fără întrerupri se tratează separat cazurile $abs(a) \le 1$ și $abs(a) > 1$.

Dacă $abs(a) \le 1$ și $x_A \le x_B$, se începe generarea din punctul $(x_1 = x_A, y_1 = y_A)$ și la fiecare pas se incrementează x cu 1, deci se iau punctele de pe dreaptă care au: $x_1 = x_A$, $x_2 = x_1+1$, $\ldots x_i$, $x_{i+1} = x_i+1, \ldots\ldots$ Dat fiind că $y_i = ax_i + b$, rezultă că

$y_{i+1} = ax_{i+1} + b = a(x_i+1) + b = y_i + a$. Deci, pentru trecerea de la un punct la următorul, x se incrementează cu 1, iar y se incrementează cu valoarea pantei. Punctului de pe dreaptă de coordonate (x_i, y_i), îi corespunde un pixel a cărui adresă în spațiul imagine se calculează prin trunchiere sau rotunjire.

Dacă abs(a) ≤ 1 şi $x_A > x_B$, se începe generarea din punctul ($x_1 = x_B$, $y_1 = y_B$) şi restul calculelor se continuă la fel ca în cazul precedent.

Dacă abs(a) >1 şi $x_A \leq x_B$, se începe generarea segmentului din punctul ($x_1 = x_A$, $y_1 = y_A$) şi la fiecare pas se incrementează y cu 1, deci se calculează punctele care au: $y_1 = y_A$, $y_2 = y_1+1$,....y_i, $y_{i+1} = y_i+1$,... Deoarece $x_i = (y_i - b)/a$, rezultă că $x_{i+1} = (y_{i+1} - b)a = x_i+1/a$. Deci, pentru trecerea de la un punct la următorul, y se incrementează cu 1, iar x se incrementează cu inversul pantei.

Dacă abs(a) > 1 şi $x_A > x_B$, se începe generarea din punctul ($x_1 = x_B$, $y_1 = y_B$) şi restul calculelor se continuă la fel ca în cazul precedent.

Algoritmul incremental de generare a primitivelor geometrice are mai multe variante de implementare, în funcție de situația în care este aplicat. Pentru segmente de dreaptă, se adaptează algoritmul incremental astfel încât să se genereze segmente fără întreruperi, folosind numai operații cu numere întregi. Această variantă este cunoscută sub numele de algoritmul Bresenham. Pentru generarea poligoanelor, varianta de algoritm incremental este algoritmul de baleiere pe linii (*scan-line*).

Algoritmul Bresenham se bazează pe metoda incrementală, dar este optimizat astfel încât foloseşte numai operații cu numere întregi. Datorită acestei proprietăți, el poate fi implementat hardware şi, în majoritatea cazurilor, operațiile de generare a segmentelor de dreaptă din acceleratoarele grafice se bazează pe o implementare hardware a algoritmului Bresenham. Descrierea detaliată a algoritmului Bresenham se poate găsi în bibliografie [Bre65], [Mold96].

Indiferent de algoritmul de conversie folosit pentru generarea liniilor, trecerea de la spațiul bidimensional continuu, în care poziția punctelor este reprezentată prin numere reale, la reprezentarea în spațiul discret al imaginii este o aproximare prin eşantionare, care produce erori de reprezentare. Din cauza acestei aproximări, orice linie dreaptă (cu excepția celor orizontale sau verticale) apare zimțată (*jagged*), ca o "scară" de segmente de dreaptă succesive, aşa cum se poate observa în orice imagine generată pe un display sau reprodusă pe imprimantă, inclusiv în textul de față. Imaginea de scară este cu atât mai pregnantă cu cât rezoluția display-ului (numărul de pixeli pe orizontală şi pe verticală) este mai mică. Dacă rezoluția crește, aspectul nedorit de scară este atenuat, dar nu este eliminat. Netezirea liniilor se poate obține prin diferite metode de filtrare, metode cunoscute sub numele de filtrare anti-aliasing.

5.2 GENERAREA POLIGOANELOR

Metoda cea mai frecvent folosită pentru generarea mulțimii pixelilor prin care se aproximează o suprafață poligonală este metoda de baleiere pe linii a

poligonului (algoritmul *scan-line*). Principiul de bază al acestei metode pentru un poligon convex este prezentat în fig. 5.3.

Se consideră poligonul A'B'C'D'E', care reprezintă proiecția în planul $z_S = 0$ al sistemului de referință ecran 3D, a unei suprafețe ABCDE din spațiul tridimensional universal. Poligonul este baleiat cu o linie orizontală care ia valorile succesive $y_1, y_2, \ldots y_i, y_{i+1}..$, unde y_1 este valoarea coordonatei y minimă a vârfurilor poligonului, iar $y_{i+1} = y_i + 1$. Pentru fiecare poziție y_i a liniei de baleiere, se calculează intersecțiile $x_{i,1}$, $x_{i,2}$ cu două din laturile poligonului. Pixelii a căror adrese sunt cuprinse în intervalul $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$, aparțin poligonului și sunt prelucrați în mod corespunzător. Pentru algoritmul simplu de generare a poligoanelor, culoarea poligonului (cunoscută ca atribut a feței corespunzătoare a obiectului) este înscrisă în bufferul de imagine la toate adresele cuprinse în intervalul respectiv. În algoritmii în care se combină generarea cu eliminarea suprafețelor ascunse sau umbrire se prelucrează toți pixelii corespunzători acestui interval.

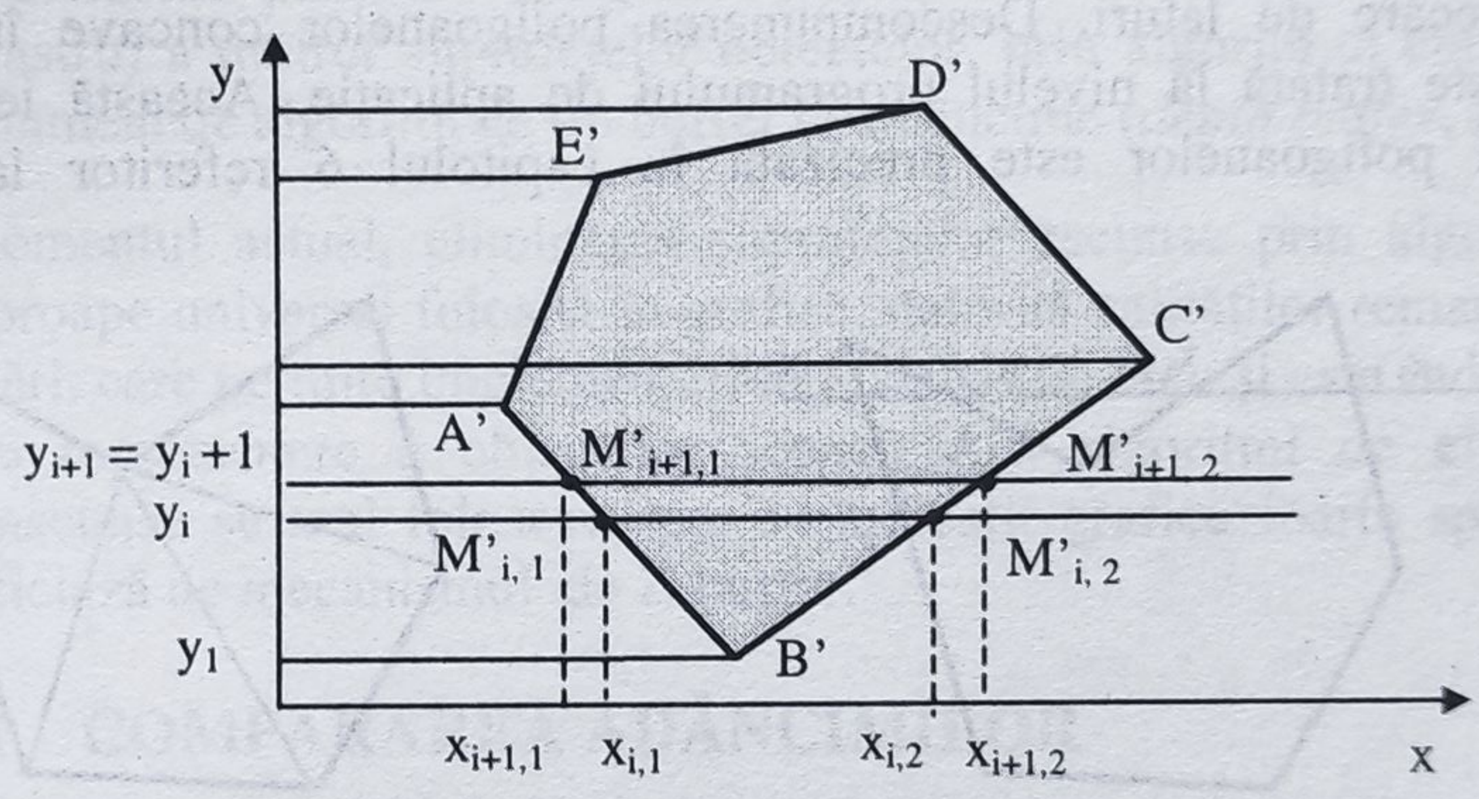


Fig. 5.3 Principiul baleierii pe linii a poligoanelor (*scan-line*).

Fie următoarele ecuații ale laturilor stânga și dreapta cu care se intersectează dreapta $y = y_i$:

$$x = m_1 y + n_1$$
$$x = m_2 y + n_2 \qquad (5.1)$$

Pentru situația din fig.5.3, intersecția dreptei $y = y_i$ are loc cu laturile A'B' și B'C', pentru care coeficienții din ecuațiile de mai sus au valorile:

$$m_1 = (x_A - x_B) / (y_A - y_B), \quad n_1 = x_B - m_1 y_B$$
$$m_2 = (x_C - x_B) / (y_C - y_B), \quad n_2 = x_C - m_2 y_C \qquad (5.2)$$

Intersecțiile corespunzătoare sunt: $x_{i,1} = m_1 y_i + n_1$, $x_{i,2} = m_2 y_i + n_2$. Coordonatele intersecțiilor liniei următoare de baleiere $y = y_{i+1}$, unde $y_{i+1} = y_i + 1$, se pot calcula din coordonatele precedente folosind coeficienții m_1 și m_2 ai laturilor intersectate:

$$x_{i+1,1} = x_{i,1} + m_1$$
$$x_{i+1,2} = x_{i,2} + m_2 \qquad (5.3)$$

Secvența de pixeli ai căror adrese sunt cuprinse în intervalul ($x_{i+1,1}$, y_{i+1}), ($x_{i+1,2}$,y_{i+1}) aparțin poligonului pe linia de baleiere $y = y_{i+1}$ şi sunt tratați corespunzător.

Acesta este principiul de bază al baleierii pe linii (*scan-line*). Cazurile particulare, când se schimbă latura cu care are loc intersecția (la colțurile poligonului), sau când există laturi orizontale, se tratează separat.

Algoritmul de baleiere pe linie se poate implementa şi pentru poligoane concave. În acest caz, o linie de baleiere poate interesecta mai multe perechi de laturi, rezultând mai multe segmente de interecție şi deci mai multe secvențe de pixeli care aparțin poligonului (fig. 5.4(a)). În general, însă, poligoanele concave sau cu găuri se tratează pe mai multe nivele. Acceleratoarele grafice implementează hardware automate de generare a unor suprafețe convexe simple (triunghiuri sau trapeze). Descompunerea unui poligon convex în mai multe triunghiuri este o operație simplă care, de cele mai multe ori, este preluată de biblioteca grafică, care conține funcții de generare a poligoanelor convexe cu un număr oarecare de laturi. Descompunerea poligoanelor concave în poligoane convexe este tratată la nivelul programului de aplicație. Această ierarhizare în prelucrarea poligoanelor este precizată în capitolul 6 referitor la biblioteca OpenGL.

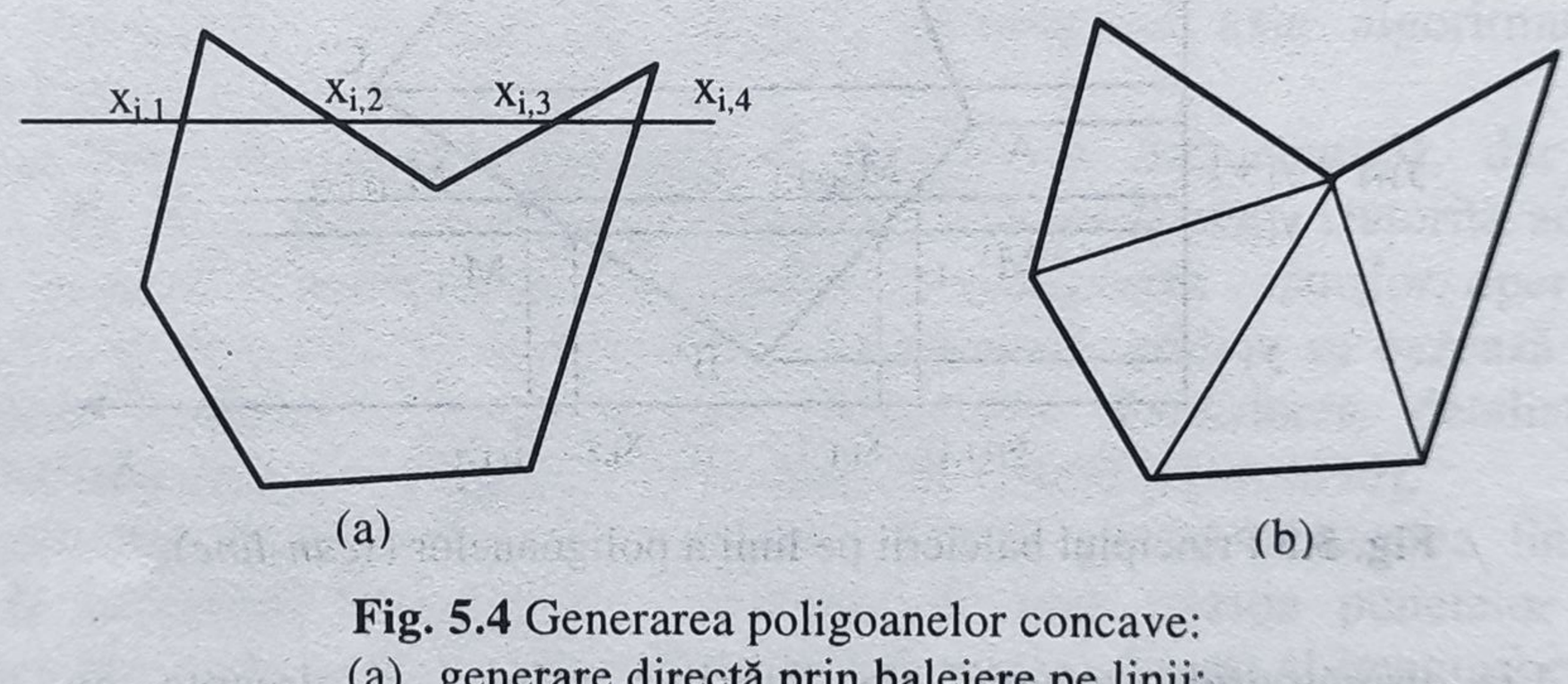


Fig. 5.4 Generarea poligoanelor concave:
(a) generare directă prin baleiere pe linii;
(b) transformare în mulțime de triunghiuri (triangularizare).

Ca şi în cazul liniilor, muchiile poligoanelor generate prezintă aspectul de scară de segmente, datorită trecerii la spațiul discret al imaginii. Algoritmii de filtrare pentru netezirea muchiilor poligonului (algoritmi de anti-aliasing) se execută, în general, concomitent cu algoritmul de generare scan-line a poligonului.

5.3 ELIMINAREA SUPRAFEȚELOR ASCUNSE

Reprezentarea pe display a tuturor suprafețelor obiectelor proiectate în fereastra de vizualizare produce o imagine nerealistă, deoarece este posibil ca o suprafață care se află în spatele altei suprafețe să fie vizibilă pe ecran. Această situație provine din transformarea de proiecție din spațiul tridimensional în spațiul

bidimensional (planul de vizualizare). În general, orice proiecţie înseamnă o reducere a dimensionalităţii spaţiului de reprezentare şi, prin acesta, se pierd o parte din proprietăţile formelor. Datorită proiecţiei, suprafeţe distincte, aflate la distanţe diferite în spaţiu, sunt proiectate în aceeaşi regiune pe display şi sunt reprezentate suprapus. Pentru obţinerea unor imagini corecte, care să reflecte relaţia spaţială dintre obiecte, se prelucrează suplimentar suprafeţele proiectate ale obiectelor.

Prelucrarea suprafeţelor din punct de vedere al poziţiei relative între ele, prelucrare cunoscută sub numele de eliminarea suprafeţelor ascunse (*hidden surface removal*) se poate efectua în două moduri:

(a) Eliminarea suprafeţelor ascunse în spaţiul obiect, prin compararea adâncimii relative a suprafeţelor obiectelor în sistemul de referinţă de observare sau în sistemul de referinţă normalizat.

(b) Eliminarea suprafeţelor ascunse în spaţiul imagine, prin compararea adâncimii pixelilor imaginii rezultaţi din generarea (conversia de rastru) a tuturor suprafeţelor obiectelor, prin algoritmul cunoscut sub numele de algoritm de tip buffer de adâncime (*depth buffer*, *Z-buffer*).

În momentul actual, eliminarea suprafeţelor ascunse prin algoritmul Z-buffer este aproape universal folosită în grafică, datorită calităţilor remarcabile ale acestei abordări, care permite implementări hardware eficiente şi este independentă de modul de reprezentare a obiectelor scenei. Alţi algoritmi de eliminare a suprafeţelor ascunse se mai folosesc doar în aplicaţii grafice foarte specializate, care nu beneficiază de mecanismul de Z-buffer.

5.3.1 COMPARAREA ADÂNCIMILOR

Pentru eliminarea suprafeţelor ascunse se compară adâncimea (distanţa faţă de punctul de observare) tuturor suprafeţelor sau a pixelilor rezultaţi prin conversia de rastru.

Un punct P este ascuns de o suprafaţă S, dacă adâncimea punctului este mai mare decât adâncimea intersecţiei I a dreaptei care uneşte punctul P cu centrul de proiecţie (se consideră centrul de proiecţie identic cu originea sistemului de referinţă de observare O_V , fig. 5.5(a)), deci dacă $O_VP > O_VI$.

O suprafaţă A este ascunsă de altă suprafaţă S dacă toate punctele suprafeţei A sunt ascunse de suprafaţa S (fig. 5.5(b)). Dacă o suprafaţă ascunde S suprafaţa A, atunci se consideră că adâncimea suprafeţei S este mai mică decât cea a suprafeţei ascunse A. Compararea adâncimilor a două puncte în sistemul de referinţă de observare este destul de complicată, deoarece comparaţia are sens numai dacă cele două puncte se află pe aceeaşi linie de proiecţie (fig. 5.5).

Comparaţia adâncimilor este mai simplă în sistemul de referinţă ecran 3D. În acest sistem, $z_S = z_N$, iar z_N este definit de relaţia (4.10) astfel:

$$z_S = z_N = \frac{f}{f-d}\left(1-\frac{d}{z_V}\right)$$

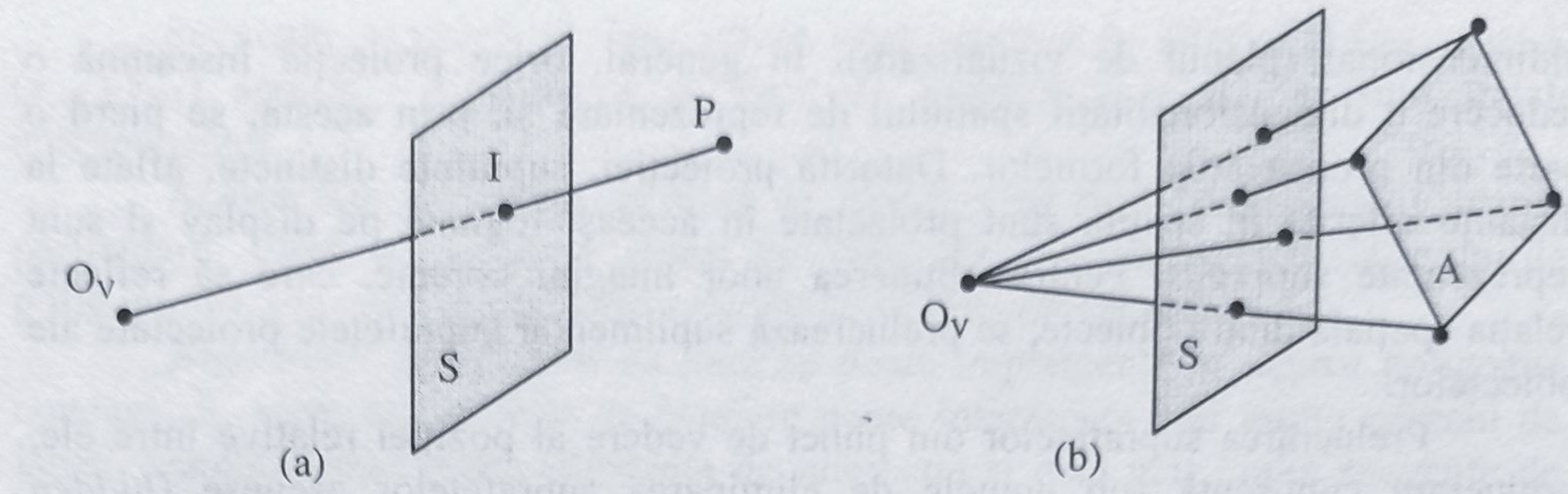


Fig. 5.5 (a) Suprafața S ascunde punctul P.
(b) Suprafața S ascunde suprafața A.

Acestă relație transformă, de exemplu, un cub în sistemul de referință de observare, într-un trunchi de piramidă în sistemul ecran 3D (fig. 5.6).

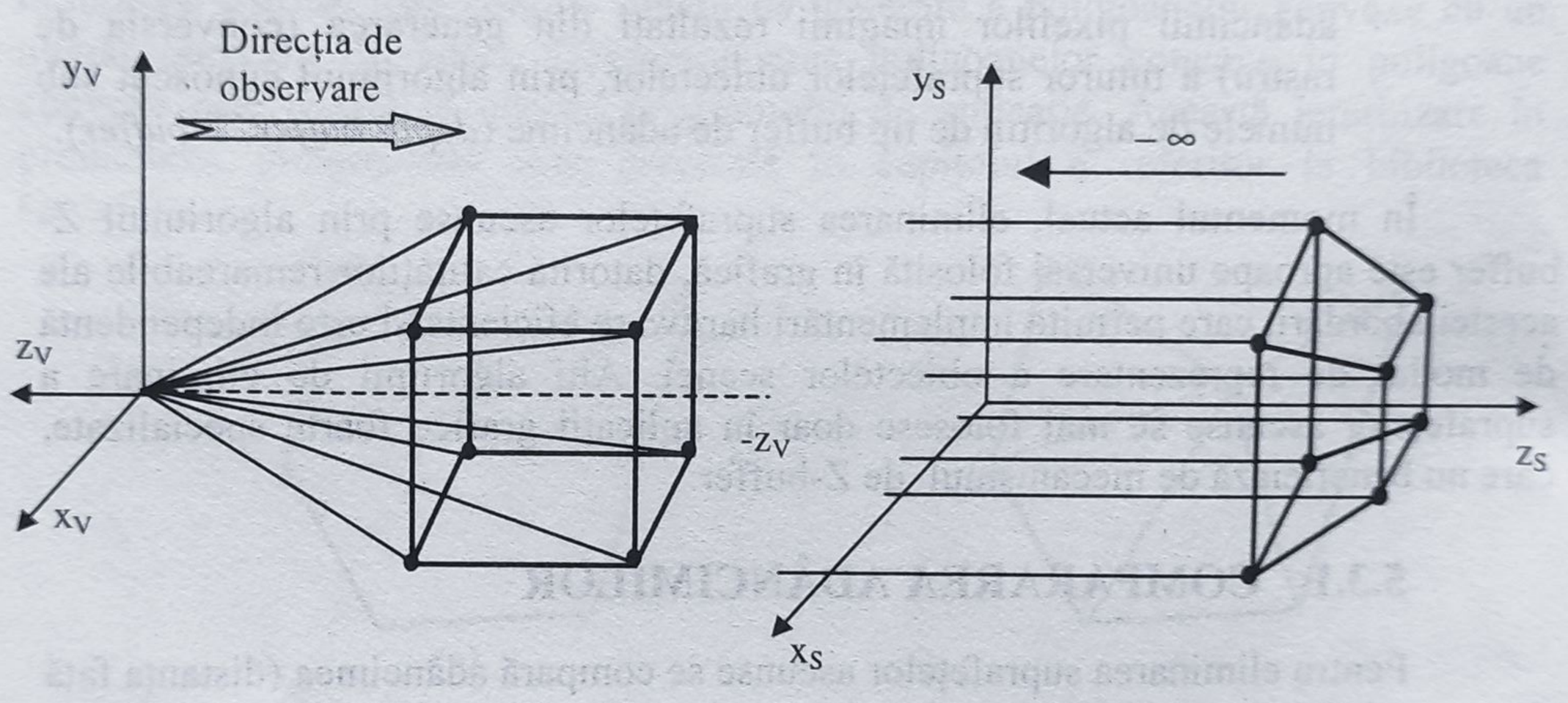


Fig. 5.6 Transformarea unui cub şi a liniilor de proiecție din sistemul de referință de observare în sistemul de referință ecran 3D.

Proiecția perspectivă din sistemul de referință de observare se transformă în proiecție paralelă în sistemul de referință ecran 3D. Deoarece $z_V = 0$ se transformă în $z_S = -\infty$, liniile de proiecție în sistemul ecran 3D sunt linii paralele îndreptate către $-\infty$. Adâncimea unui punct $P_1(x_1,y_1,z_1)$ în sistemul de referință de observare, care este dată de distanța $O_V P_1 = \sqrt{x_1^2 + y_1^2 + z_1^2}$, se transformă în coordonata z_S a punctului în sistemul de referință ecran 3D.

În sistemul de referință de observare, testul de adâncime pentru a determina dacă un punct P_1 este mai aproape de centrul de proiecție (care este, în general, acelaşi cu punctul de observare) decât punctul P_2, se poate efectua numai dacă cele două puncte se află pe aceeaşi linie de proiecție. În sistemul ecran 3D, toate punctele care au aceleaşi coordonate (x_S, y_S) în planul de proiecție se află pe acelaşi proiector, o dreaptă paralelă cu axa z_S, deci se poate efectua testul de adâncime prin comparația coordonatelor z_S ale punctelor.

Se poate trage concluzia că operaţiile care implică testarea sau compararea adâncimilor se execută mai simplu în sistemul de referinţă ecran 3D. Acest lucru este utilizat atât în algoritmii de eliminare a suprafeţelor ascunse în spaţiul obiect, cât şi în algoritmii în spaţiul imagine.

5.3.2 ELIMINAREA SUPRAFEȚELOR ASCUNSE ÎN SPAȚIUL OBIECT

Unul din cei mai cunoscuţi algoritmi de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul propus de Newell, cunoscut şi sub numele de *algoritmul pictorului*, datorită analogiei cu modul în care un pictor creează un tablou [New72]. În acest algoritm, se sortează suprafeţele în ordinea descrescătoare a adâncimii şi se redau în această ordine pe display. Eliminarea suprafeţelor ascunse se obţine prin suprapunerea noii suprafeţe (mai apropiată) peste suprafeţele mai depărtate, deja desenate. Lista ordonată de suprafeţe (numită listă de display) se creează în sistemul de referinţă de observare sau în sistemul de referinţă ecran 3D, după operaţia de decupare.

Ordonarea suprafeţelor după adâncime este o operaţie complicată, mai ales dacă două suprafeţe se intersectează între ele, situaţie în care nu se poate decide care suprafaţă este mai apropiată, decât prin divizarea uneia din ele de-a lungul liniei de intersecţie.

Un alt algoritm de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul cu subdivizarea recursivă a poligoanelor (algoritmul Weiler-Atherton, [Wei77]). Spre deosebire de algoritmul pictorului, în care intersectarea suprafeţelor proiectate se efectuează prin supraînscrierea suprafeţelor mai apropiate peste cele mai depărtate, în algoritmul Weiler-Atherton se execută o operaţie de decupare în sistemul de referinţă normalizat între toate suprafeţele obiectelor şi se reţin numai suprafeţele "învingătoare", adică cele mai apropiate. Lista acestora (lista de display) nu mai conţine suprafeţe care se pot suprapune pe ecran şi nu mai trebuie să fie ordonată. Ca şi în algoritmul pictorului, operaţiile de creare a listei de display necesită un timp de execuţie foarte mare, care depinde de numărul de suprafeţe ce se prelucrează. Acest lucru limitează drastic posibilitatea de utilizare a algoritmilor în spaţiul obiect.

5.3.3 ELIMINAREA SUPRAFEȚELOR ASCUNSE ÎN SPAȚIUL IMAGINE: ALGORITMUL Z-BUFFER

Dezvoltat de Catmull în anul 1975 [Cat75], algoritmul Z-buffer introduce o metodă simplă de eliminare a suprafeţelor ascunse, în care se efectuează comparaţia între adâncimea z_S a fiecărui pixel (x, y) al poligoanelor generate şi se selectează pixelul cu adâncimea minimă (cel mai apropiat).

Această selecţie se implementează cel mai convenabil prin folosirea unui buffer de adâncime (Z-buffer), care memorează adâncimea tuturor pixelilor generaţi în spaţiul de imagine, a căror culoare este memorată în bufferul de imagine. În orice moment, la fiecare adresă (x, y) în Z-buffer, este memorată

Se poate trage concluzia că operaţiile care implică testarea sau compararea adâncimilor se execută mai simplu în sistemul de referinţă ecran 3D. Acest lucru este utilizat atât în algoritmii de eliminare a suprafeţelor ascunse în spaţiul obiect, cât şi în algoritmii în spaţiul imagine.

5.3.2 ELIMINAREA SUPRAFEŢELOR ASCUNSE ÎN SPAŢIUL OBIECT

Unul din cei mai cunoscuţi algoritmi de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul propus de Newell, cunoscut şi sub numele de *algoritmul pictorului*, datorită analogiei cu modul în care un pictor creează un tablou [New72]. În acest algoritm, se sortează suprafeţele în ordinea descrescătoare a adâncimii şi se redau în această ordine pe display. Eliminarea suprafeţelor ascunse se obţine prin suprapunerea noii suprafeţe (mai apropiată) peste suprafeţele mai depărtate, deja desenate. Lista ordonată de suprafeţe (numită listă de display) se creează în sistemul de referinţă de observare sau în sistemul de referinţă ecran 3D, după operaţia de decupare.

Ordonarea suprafeţelor după adâncime este o operaţie complicată, mai ales dacă două suprafeţe se intersectează între ele, situaţie în care nu se poate decide care suprafaţă este mai apropiată, decât prin divizarea uneia din ele de-a lungul liniei de intersecţie.

Un alt algoritm de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul cu subdivizarea recursivă a poligoanelor (algoritmul Weiler-Atherton, [Wei77]). Spre deosebire de algoritmul pictorului, în care intersectarea suprafeţelor proiectate se efectuează prin supraînscrierea suprafeţelor mai apropiate peste cele mai depărtate, în algoritmul Weiler-Atherton se execută o operaţie de decupare în sistemul de referinţă normalizat între toate suprafeţele obiectelor şi se reţin numai suprafeţele "învingătoare", adică cele mai apropiate. Lista acestora (lista de display) nu mai conţine suprafeţe care se pot suprapune pe ecran şi nu mai trebuie să fie ordonată. Ca şi în algoritmul pictorului, operaţiile de creare a listei de display necesită un timp de execuţie foarte mare, care depinde de numărul de suprafeţe ce se prelucrează. Acest lucru limitează drastic posibilitatea de utilizare a algoritmilor în spaţiul obiect.

5.3.3 ELIMINAREA SUPRAFEŢELOR ASCUNSE ÎN SPAŢIUL IMAGINE: ALGORITMUL Z-BUFFER

Dezvoltat de Catmull în anul 1975 [Cat75], algoritmul Z-buffer introduce o metodă simplă de eliminare a suprafeţelor ascunse, în care se efectuează comparaţia între adâncimea z_S a fiecărui pixel (x, y) al poligoanelor generate şi se selectează pixelul cu adâncimea minimă (cel mai apropiat).

Această selecţie se implementează cel mai convenabil prin folosirea unui buffer de adâncime (Z-buffer), care memorează adâncimea tuturor pixelilor generaţi în spaţiul de imagine, a căror culoare este memorată în bufferul de imagine. În orice moment, la fiecare adresă (x, y) în Z-buffer, este memorată

adâncimea celui mai apropiat pixel de adresă (x, y) rezultat în transformarea de rastru a poligoanelor. Adâncimea unui nou pixel generat este comparată cu adâncimea pixelului cu aceeaşi adresă din Z-buffer şi noul pixel înlocuieşte pixelul anterior numai dacă adâncimea lui este mai mică. Înlocuirea pixelului anterior cu un pixel nou generat se realizează prin înscrierea culorii noului pixel în bufferul de imagine la adresa (x, y) şi înscrierea adâncimii lui la aceeaşi adresă (x, y) în Z-buffer.

Comparaţia între adâncimile punctelor în sistemul de referinţă ecran 3D are avantajul că poate fi efectuată prin comparaţia coordonatelor z_S, dat fiind că toate punctele cu aceleaşi coordonate (x, y) se află pe aceeaşi linie de proiecţie paralelă cu axa z_S (fig. 5.8). Dar această simplitate de calcul are, totuşi, un inconvenient, şi anume acela că adâncimile din sistemul de referinţă de observare nu se transformă liniar în sistemul de referinţă ecran 3D. Intervale egale în z_V nu se transformă în intervale egale în z_S, ceea ce se poate observa şi din relaţiile de transformare (4.10). Cu cât z_V se apropie de planul de vizibilitate depărtat (*far*), z_S se apropie mai rapid de 1. Obiectelele sunt comprimate în funcţie de coordonata lor z_V, şi acest lucru are consecinţe asupra preciziei în comparaţia adâncimilor.

Algoritmul Z-buffer se execută concomitent cu transformarea de rastru a poligoanelor. Fiecare poligon este descris prin lista vârfurilor sale reprezentate în sistemul de referinţă ecran 3D. Adâncimea pixelilor poligonului se calculează prin interpolare pornind de la adâncimea vârfurilor acestuia.

Interpolarea pentru calculul adâncimii pixelilor se efectuează în două etape: o interpolare de-a lungul laturilor poligonului, prin care se obţine adâncimea pixelilor de pe conturul poligonului, şi o interpolare de-a lungul fiecărei liniei de baleiere, prin care se obţine adâncimea tuturor pixelilor interiori. Algoritmul de generare a poligoanelor prin baleiere pe linii (scan-line) se transformă astfel într-un algoritm combinat de baleiere şi Z-buffer (*scan-line Z-buffer*).

Se reia algoritmul de generare a poligoanelor prin baleiere pe linii din subcapitolul 5.2. Prin proiecţia suprafeţei ABCDE din spaţiu în planul $x_S = 0$ se obţine poligonul A''B''C''D''E'' (fig. 5.7). Liniile de baleiere ale acestui poligon sunt dreptele de ecuaţii $y = y_1$, $y = y_2$,....$y = y_i$, $y = y_{i+1}$, ...Dreapta $y = y_i$ interesctează laturile A''B'' şi B''C'' în punctele $M''_{i,1}(y_i, z_{i,1})$ şi $M''_{i,2}(y_i, z_{i,2})$.

Ecuaţiile laturilor A''B'' şi B''C'' se pot scrie similar cu relaţiile (5.1):

$$z = m_{1z}y + n_{1z}$$
$$z = m_{2z}y + n_{2z} \qquad (5.4)$$

unde: $m_{1z} = (z_A - z_B) / (y_A - y_B)$, $n_{1z} = y_B - m_{1z} z_B$

$$m_{2z} = (z_C - z_B) / (y_C - y_B), \quad n_{2z} = y_C - m_{1z} z_C \qquad (5.5)$$

Intersecţiile dreptelor A''B'' şi B''C'' cu dreapta $y = y_i$ au coordonatele $z_{i,1} = m_{1z}y_i + n_{1z}$ şi $z_{i,2} = m_{2z}y_i + n_{2z}$. Coordonatele punctelor de intersecţie ale dreptei $y = y_{i+1} = y_i + 1$, se pot calcula din coordonatele intersecţiilor precedente folosind coeficienţii m_{1z} şi m_{2z} ai laturilor intersectate:

$$z_{i+1,1} = z_{i,1} + m_{1z}$$
$$z_{i+1,2} = z_{i,2} + m_{2z} \qquad (5.6)$$

Pentru fiecare linie de baleiere se calculează coordonatele intersecţiilor cu laturile poligonului prin incrementarea valorilor precedente cu coeficienţii daţi de relaţiile (5.5).

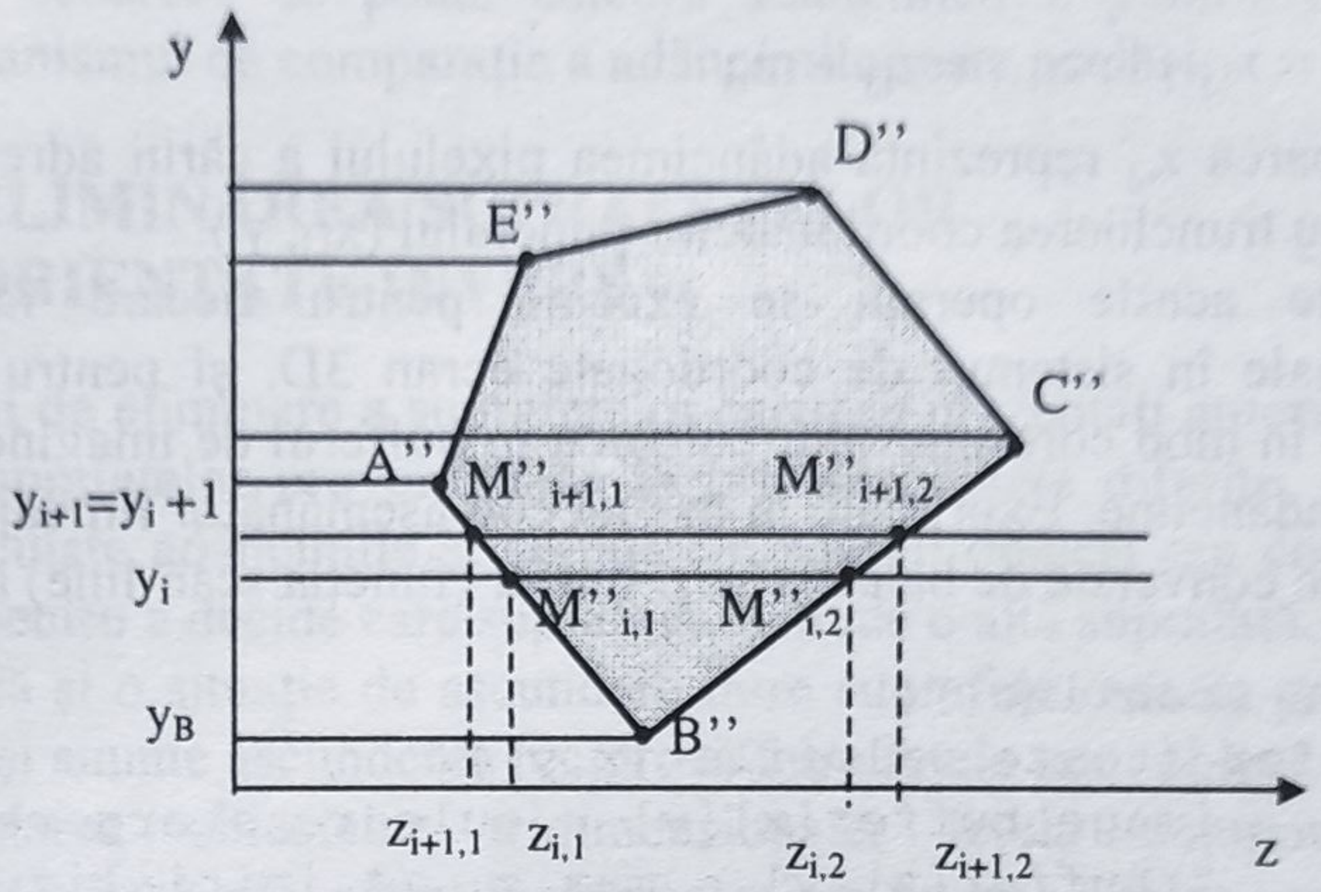


Fig. 5.7 Baleierea poligonului în planul yz.

Cele două operaţii de baleiere, pentru calculul adresei şi al adâncimii pixelilor de pe conturul unei suprafeţei ABCDE, se execută concomitent prin intersecţia suprafeţei cu planele de ecuaţii $y = y_1$, $y = y_2$,.... y_i, y_{i+1}, ... Laturile AB şi BC sunt intersectate de planul $y = y_i$ în punctele $M_{i,1}$ şi $M_{i,2}$. Intersecţiile $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$ se calculează în planul $z_S = 0$, prin intersecţia dreptei $y = y_i$ cu proiecţiile A'B' şi B'C' ale laturilor AB şi BC în planul $z_S = 0$. Intersecţiile $(z_{i,1}, y_i)$, $(z_{i,2}, y_i)$ se calculează în planul $x_S = 0$, prin intersecţia dreptei $y = y_i$ cu proiecţiile A''B'' şi B''C'' ale laturilor AB şi BC în planul $x_S = 0$.

Adresele pixelilor se obţin prin rotunjirea (sau trunchierea) coordonatelor punctelor de intersecţie $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$; valorile $z_{i,1}$ şi $z_{i,2}$ reprezintă adâncimea acestor pixeli, care se află pe conturul poligonului.

Pentru calculul adâncimii pixelilor interiori (a celor ce nu se află pe conturul poligonului), se efectuează o interpolare pe fiecare linie de baleiere.

În planul $y = y_i$, relaţia dintre coordonatele x şi z ale dreptei $M_{i,1}M_{i,2}$ este:

$$z = m_{ix}x + n_{ix} \tag{5.7}$$

unde: $m_{ix} = (x_{i,2} - x_{i,1}) / (z_{i,2} - z_{i,1})$, $n_{ix} = z_{i,1} - m_{ix}\, x_{i,1}$

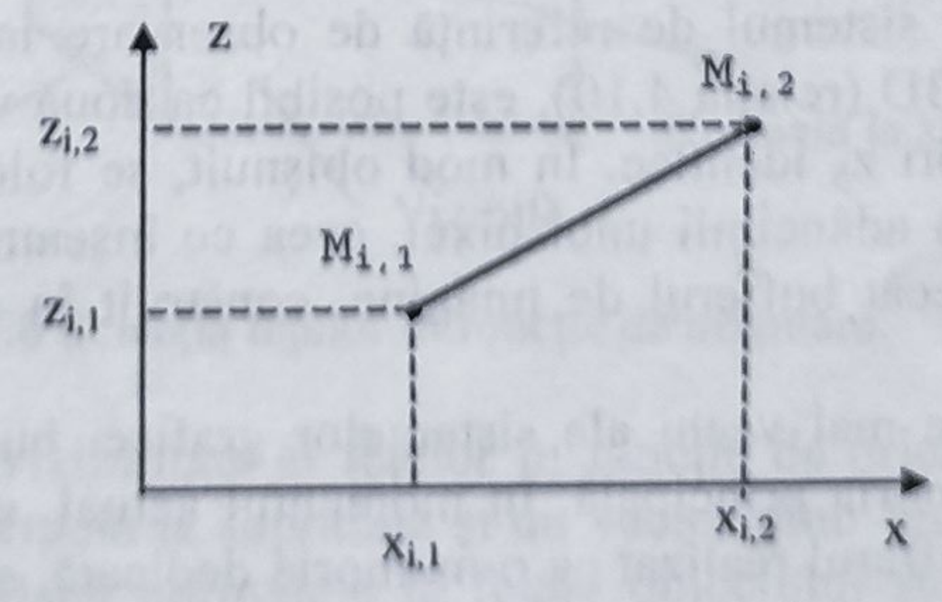


Fig. 5.8 Interpolarea adâncimilor pe linia de baleiere $y = y_i$.

Un punct pe această dreaptă are coordonatele $x_{i,j}$, $z_{i,j} = x_{i,j}\, m_{ix} + n_{ix}$. Pe segmentul de dreaptă $M_{i,1}M_{i,2}$ se calculează puncte succesive prin incrementarea cu 1 a coordonatei x şi calculul prin incrementare cu panta m_{ix} a coordonatei z:

$$x_{i,j+1} = x_{i,j} + 1;\ z_{i,j+1} = z_{i,j} + m_{ix} \tag{5.8}$$

Valoarea $z_{i,j}$ reprezintă adâncimea pixelului a cărui adresă se obține prin rotunjirea sau trunchierea coordonatelor punctului $(x_{i,j}, y_i)$.

Toate aceste operații se execută pentru fiecare față a obiectelor tridimensionale în sistemul de coordonate ecran 3D, şi pentru fiecare pixel se actualizează în mod corespunzător culoarea în bufferul de imagine şi adâncimea în bufferul de adâncime. Exprimate în pseudocod asemănător limbajului C, operațiile combinate de conversie de baleiere şi Z-buffer (funcția scan-line) arată astfel:

```
void scanline(){
   for (toate valorile (x,y)){
      image_buffer[x][y] = culoare_stergere;
      Z_buffer[x][y] = max_z; /* initializ. Z-buffer */
   }
   for (fiecare poligon){
      calculul ymin, ymax ale poligonului;
      for (yi = ymin; yi<= ymax;yi++){
         calcul xi,1, xi,2
         calcul zi,1, zi,2
         for (x=xi,1;x<xi,2;x++){
            calcul z;
            if(z < Z_buffer[x][y]){
               Z_buffer[x][y] = z;
               image_buffer[x][y] = culoare;
            }
         }
      }
   }
}
```

Cel mai important avantaj al algoritmului Z-buffer este simplitatea implementării lui, iar dezavantajul principal îl constitue memoria suplimentară necesară pentru implementarea bufferului de adâncime. Dimensiunea fiecărei locații (fiecare locație corespunzând unui pixel din imagine) este dată de precizia de reprezentare a adâncimii z. Datorită compresiei pe care o introduce transformarea de la sistemul de referință de observare la sistemele de referință normalizat şi ecran 3D (relația 4.10), este posibil ca două valori z_V distincte să fie transformate în valori z_S identice. În mod obişnuit, se folosesc 20 sau 32 de biți pentru reprezentarea adâncimii unui pixel, ceea ce înseamnă că Z-bufferul poate deveni mai mare decât bufferul de imagine, construit în mod obişnuit cu 24 de biți/pixel.

În realizările mai vechi ale sistemelor grafice, bufferul de adâncime era implementat în memoria principală. În momentul actual, sunt disponibile sisteme grafice care au Z-bufferul realizat ca o memorie dedicată, şi aceasta reprezintă cea mai bună soluție.

Un alt avantaj al Z-bufferului îl reprezintă independenţa de forma de reprezentare a obiectelor. Descrierea dată mai sus se referă la utilizarea Z-bufferului în redarea obiectelor modelate prin suprafeţe poligonale, dar aceasta nu este o restricţie, deoarece se poate calcula adâncimea z pentru orice tip de suprafaţă, iar mecanismul de comparaţie a adâncimilor este acelaşi.

5.3.4 ELIMINAREA SUPRAFEŢELOR ORIENTATE INVERS

Algoritmii de eliminare a suprafeţelor ascunse prezentaţi anterior se ocupă de compararea suprafeţelor provenind de la oricâte obiecte diferite, pentru care trebuie să fie calculate adâncimile şi sortate (în spaţiul obiect) sau comparate (în spaţiul imagine) pentru a decide care suprafaţă ascunde o altă suprafaţă.

Există însă şi o situaţie de ascundere între suprafeţe care se poate rezolva mult mai simplu, şi anume ascunderea reciprocă între feţele unui obiect opac.

În modelarea obiectelor tridimensionale, feţele componente sunt reprezentate orientat în spaţiu, astfel încât normalele tuturor feţelor unui obiect să fie îndreptate către aceeaşi regiune a spaţiului (spre interiorul sau spre exteriorul obiectului).

Dacă un obiect opac ale cărui normale la feţe sunt îndreptate spre exterior, este privit dintr-un punct de observare, feţele ale căror normale sunt orientate către punctul de observare ascund (maschează) feţele ale căror normale sunt orientate în direcţie inversă.

Orientarea consistentă a feţelor unui obiect permite selecţia numai a acelor feţe ale obiectului care sunt orientate către punctul de observare. Această operaţie de selecţie (numită eliminarea feţelor orientate invers - "din spate" - *backface elimination*) se efectuează în sistemul de referinţă de observare (fig. 5.9).

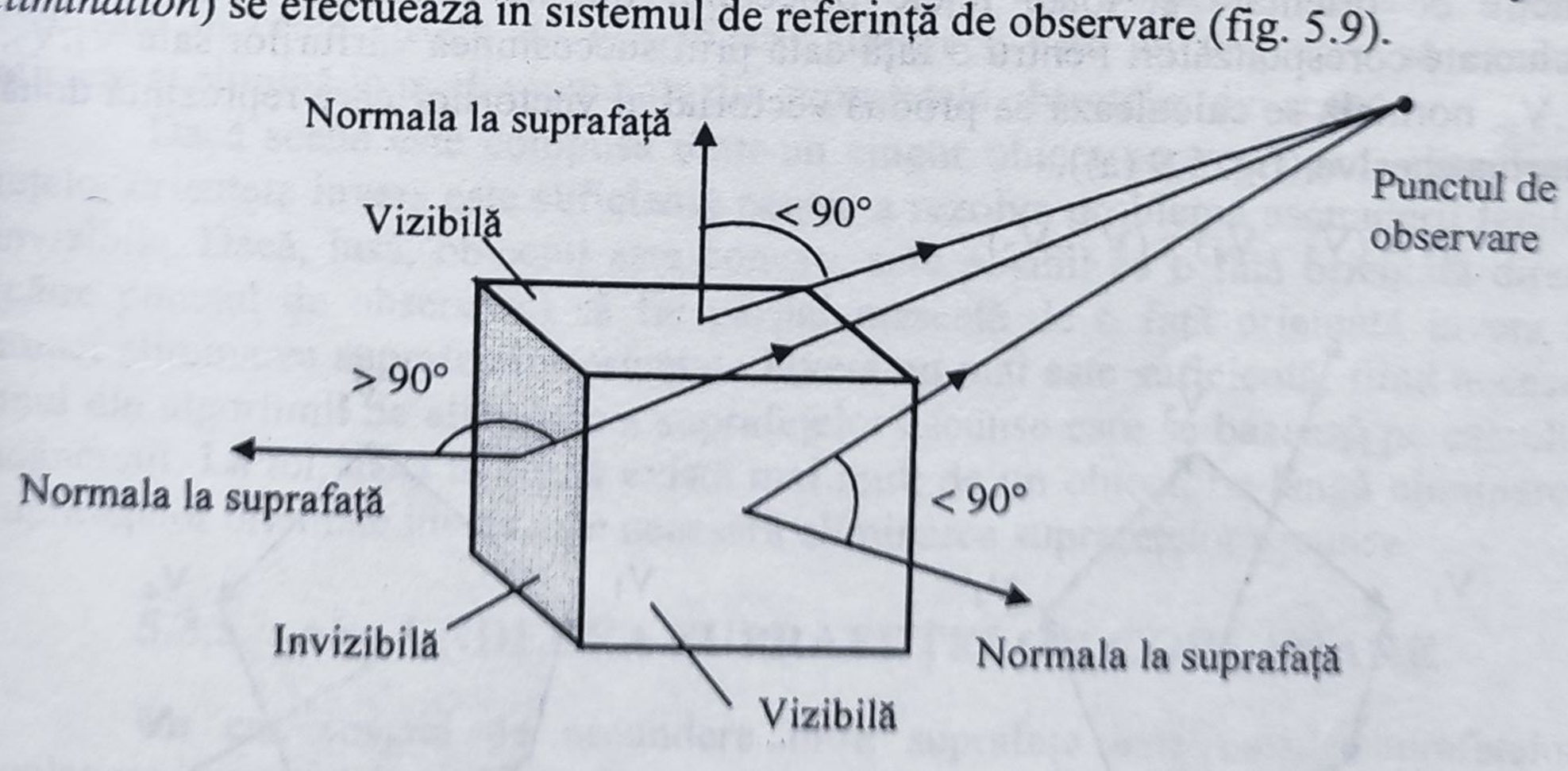


Fig. 5.8 Selecţia feţelor în funcţie de orientare.

Pentru testul de vizibilitate al feţelor în funcţie de orientare se calculează produsul scalar dintre normala la suprafaţă şi un vector dintr-un punct al feţei spre punctul de observare. Dacă normalele la feţele obiectului sunt îndreptate către

exterior (sens pozitiv, invers acelor de ceas) şi unghiul dintre cei doi vectori este mai mic de 90°, atunci suprafaţa este vizibilă.

Produsul scalar a doi vectori, **N** şi **V** este:

$$\mathbf{V} \bullet \mathbf{N} = |\mathbf{V}||\mathbf{N}|\cos\theta$$

unde |**V**| şi |**N**| sunt modulele celor doi vectori, iar θ este unghiul dintre ei. Se poate scrie deci:

$V \bullet N > 0$ dacă $\theta < 90°$
$V \bullet N = 0$ dacă $\theta = 90°$
$V \bullet N < 0$ dacă $\theta > 90°$

De aici rezultă condiţia de vizibilitate a unei feţe: pentru ca o faţă să fie vizibilă, produsul scalar dintre normala acesteia şi un vector dintr-un punct al feţei spre punctul de observare trebuie să fie pozitiv:

$$\mathbf{N} \bullet \mathbf{V} > 0 \qquad (5.9)$$

Dacă se consideră normala la suprafaţă orientată către interiorul obiectului (în sens negativ, în direcţia acelor de ceas), atunci condiţia de vizibilitate a unei feţei este ca produsul scalar dintre normală şi un vector dintr-un punct al feţei spre punctul de observare să fie negativ.

Eliminarea feţelor orientate invers se execută în sistemul de referinţă de observare; orice faţă care este invizibilă este ignorată pentru operaţiile ulterioare de redare. Acest test este foarte important deoarece, în medie, jumătate din feţele unui obiect tridimensional sunt orientate invers faţă de punctul de observare şi pot fi eliminate din calcul.

Dacă obiectul are feţe transparente, atunci nu se pot elimina feţele în funcţie de orientare şi toate feţele obiectului trebuie luate în considerare şi prelucrate corespunzător. Pentru o faţă dată prin succesiunea vârfurilor sale $V_1, V_2, \ldots V_n$, normala se calculează ca produs vectorial al vectorilor care reprezintă două laturi succesive (fig. 5.9 (a)).

$$\mathbf{N} = (V_2 - V_1) \times (V_3 - V_2)$$

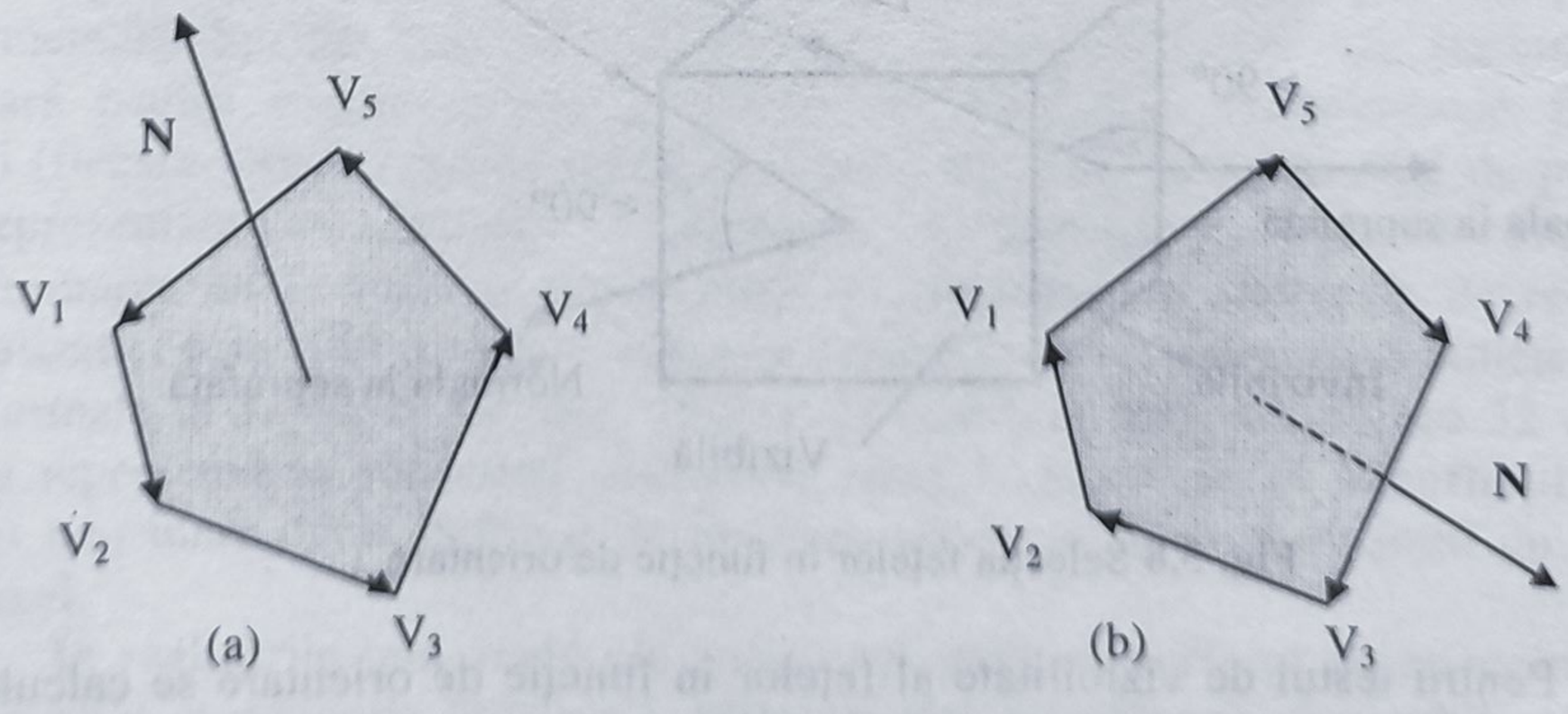


Fig. 5.9 Calculul normalei la o suprafaţă.

Dacă (x_m, y_m, z_m) reprezintă coordonatele vârfului V_m, iar **i**, **j**, **k** sunt versorii axelor x, y, z, atunci normala **N** are expresia:

$$\mathbf{N} = (V_2 - V_1) \times (V_3 - V_2) = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \end{vmatrix}$$

Se dezvoltă determinantul după prima linie, şi rezultă:

$$\begin{aligned} \mathbf{N} = &((y_2 - y_1)(z_3 - z_2) - (y_3 - y_2)(z_2 - z_1))\,\mathbf{i} + \\ &((x_3 - x_2)(z_2 - z_1) - (x_2 - x_1)(z_3 - z_2))\,\mathbf{j} + \\ &((x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1))\,\mathbf{k} \end{aligned} \qquad (5.10)$$

Dacă ordinea de parcurgere a vârfurilor feţei se inversează, deci se consideră faţa $V_n, \ldots V_2V_1$, atunci normala se calculează prin produsul $\mathbf{N}=(V_2-V_3)\times(V_1-V_2)$ care are semnul inversat (fig. 5.10 (b)), ceea ce se regăseşte în formula de calcul a produsului vectorial.

Rezultă că sensul vectorului normal la o faţă este dat de ordinea de parcurgere a vârfurilor feţei. Dacă în sistemul de referinţă local (de modelare) feţele unui obiect sunt definite (prin ordinea vârfurilor) astfel încât toate normalele să fie îndreptate spre aceeaşi regiune a spaţiului (spre exteriorul obiectului), această orientare se păstrează după transformarea de instanţiere şi observare, şi în sistemul de referinţă de observare obiectele vor avea normalele orientate consistent (toate orientate către exterior). Această condiţie este suficientă pentru calculul vizibilităţii fiecărei feţe prin produs scalar al normalei cu vectorul de direcţie de la un punct al feţei către punctul de observare.

Eliminarea suprafeţelor orientate invers ale obiectelor opace trebuie efectuată în orice sistem de vizualizare, deoarece este un test care se execută eficient şi elimină în medie jumătate din suprafeţele obiectelor din scenă.

Dacă scena este compusă dintr-un singur obiect convex opac, eliminarea feţelor orientate invers este suficientă pentru a rezolva problema ascunderii feţelor invizibile. Dacă, însă, obiectul este concav, este posibil ca o faţă orientată direct (către punctul de observare) să fie parţial mascată de o faţă orientată invers şi atunci eliminarea suprafeţelor orientate invers nu mai este suficientă, fiind necesar unul din algoritmii de eliminare a suprafeţelor ascunse care se bazează pe calculul adâncimii. La fel, dacă în scenă există mai mult de un obiect, pe lângă eliminarea suprafeţelor orientate invers este necesară eliminarea suprafeţelor ascunse.

5.3.5 ASCUNDEREA SUPRAFEŢELOR COPLANARE

Un caz special de ascundere între suprafeţe este cazul suprafeţelor coplanare. Suprafeţele coplanare sunt necesare în modelarea obiectelor atunci când pe o suprafaţă dată a unui obiect trebuie să fie suprapusă o altă suprafaţă. Astfel de situaţii se întâlnesc în reprezentarea pistelor din aeroporturi sau reprezentarea marcajelor aplicate diferitelor obiecte.

În algoritmii de eliminare a suprafețelor ascunse în spațiul obiect se poate impune condiția ca suprafața aplicată să fie introdusă în lista de display după suprafața suport şi în acest fel suprafața aplicată să fie vizibilă.

În algoritmii de eliminare a suprafețelor ascunse în spațiul imagine nu se poate selecta care suprafață este vizibilă dintr-un punct de observare dat, deoarece adâncimile a două suprafețe coplanare sunt egale în toate punctele comune. De aceea sunt necesare tehnici speciale pentru selectarea suprafeței vizibile.

Una din posibilități este de a modifica poziția suprafeței aplicate prin translatare cu un vector de translație perpendicular pe suprafața suport şi valoare a modulului astfel aleasă încât să permită discriminarea adâncimii şi funcționarea corectă a algoritmului de Z-buffer.

O altă posibilitate este folosirea unui buffer şablon (*stencil buffer*). Un buffer şablon este un buffer de dimensiune egală cu bufferul de imagine ca număr de pixeli reprezentabili. În fiecare locație a bufferului şablon se înscrie o valoare care modifică comportarea algoritmului Z-buffer: un pixel nou calculat înlocuiește pixelul existent în bufferul de imagine nu în funcție de adâncime (memorată în poziția corespuntătoare în Z-buffer), ci în funcție de valoarea memorată în bufferul şablon. În biblioteca OpenGL este prevăzut un buffer şablon care poate fi utilizat pentru redarea suprafețelor coplanare.

6

BIBLIOTECA GRAFICĂ OPENGL

Pentru programarea aplicațiilor grafice complexe necesare în crearea și redarea scenelor virtuale, se pot utiliza în momentul de față mai multe biblioteci și interfețe grafice, precum şi sisteme de dezvoltare de programe (toolkit-uri), care permit proiectantului să reutilizeze un număr mare de funcții grafice deja implementate şi să-şi concentreze eforturile asupra aplicației însăşi. Dat fiind că majoritatea acestor biblioteci sunt foarte iefine sau accesibile gratis prin Internet și pot fi folosite într-un număr mare de platforme hardware şi software, cunoaşterea și utilizarea lor este deosebit de importantă şi utilă.

Dintre bibliotecile grafice existente, biblioteca OpenGL, scrisă în limbajul C, este una dintre cele mai utilizate, datorită faptului că implementează un mare număr de funcții grafice de bază pentru crearea aplicțiilor grafice interactive, asigurând o interfață independentă de platforma hardware.

Introdusă în anul 1992, biblioteca OpenGL a devenit în momentul de față cea mai intens folosită interfață grafică pentru o mare varietate de platforme hardware şi pentru o mare varietate de aplicații grafice 2D şi 3D, de la proiectare asistată de calculator (CAD), la animație, imagistică medicală şi realitate virtuală. Datorită suportului oferit prin specificațiile unui consorțiu specializat (OpenGL Architecture Review Board) şi numărului mare de implementări existente, în momentul de față OpenGL este în mod real singurul standard grafic multiplatformă. Calitățile de maturitate, stabilitate, portabilitate şi fiabilitate se adaugă la cele de compatibilitate între versiunile existente şi de uşurință de utilizare datorită unei bune structurări şi documentației care se găseşte cu uşurință. Au fost publicate numeroase cărți, exemple de cod şi alte informații ieftine sau accesibile liber pe rețeaua Internet. Una din adresele Internet care se pot folosi pentru astfel de informații este www.opengl.org.

Aplicațiile OpenGL pot rula pe platforme foarte variate, începând cu PC, stații de lucru şi până la supercalculatoare, sub cele mai cunoscute sisteme de operare: Linux, Unix, Windows NT, Windows 95, Mac OS. De asemenea se integrează în orice sistem de administrare a ferestrelor (windowing), incluzând Win32, X/Windows, şi Presentation Manager. Funcțiile OpenGL sunt apelabile din

limbajele Ada, C, C++ şi Java. Ca rezultat, aplicaţiile scrise pentru OpenGL pot fi adaptate pentru orice sistem disponibil al utilizatorilor. Din consorţiul de arhitectură OpenGL fac parte reprezentanţi de la firmele Compaq, Evens-Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Microsoft şi Silcon Graphics. Pentru scrierea aplicaţiilor folosind interfaţa OpenGL nu este necesară obţinerea unei licenţe de către utilizatori finali.

Implementările bibliotecii OpenGL diferă de la un sistem grafic la altul, dezvoltatorii sistemelor având astfel posibilitatea de optimiza costul şi performaţele acestora. Funcţiile OpenGL, care sunt aceleaşi ca interfaţă de apel, pot fi executate de hardware specializat, pot fi executate ca rutine soft de procesorul unităţii centrale CPU, sau pot fi implementate ca o combinaţie de rutine software şi hardware specializat. Această flexibilitate de implementare a permis realizarea unei game variate de acceleratoarele grafice OpenGL, de la simple plăci de redare grafică, până la implemetarea completă a pipeline-ului grafic, inclusiv funcţiile de Z-buffer, anti-alising, ceaţă, texturare. Există acceleratoare grafice OpenGL de la plăci cu preţ scăzut pentru calculatoare PC, până la sisteme grafice ale staţiilor de lucru puternice şi multiprocesoare.

În lucrarea de faţă sunt prezentate aspectele cele mai importante ale programării aplicaţiilor grafice folosind biblioteca OpenGL, cu intenţia de a asigura un minim de cunoştinţe de pornire a experienţei de programare. De asemenea, sunt descrise programele folosite pentru exemplificarea aspectelor teoretice de modelare şi generare a imaginilor (prezentate în capitolele 2, 3, 4 şi 5). Prezentarea completă a tuturor funcţiilor bibliotecii OpenGL nu este nici posibilă, nici necesară. Numai ghidul de programare OpenGL (*OpenGL Programming Guide* [Woo97]), conţine peste 600 de pagini, la care se adaugă manualele de referinţă şi multe altele. Dar, odată înţelese principiile de organizare a interfeţei, restul detaliilor de implementare se pot parcurge cu uşurinţă din orice material disponibil sau help on-line.

Folosind biblioteca OpenGL, un programator în domeniul graficii se poate concentra asupra aplicaţiei dorite şi nu mai trebuie să fie preocupat de detaliile de implementare a diferitelor funcţii "standard". Programatorul nu este obligat să scrie algoritmul de generare a liniilor sau suprafeţelor, nu trebuie să calculeze decuparea obiectelor la volumul de vizualizare etc., toate acestea fiind deja implementate în funcţiile bibliotecii şi, cel mai probabil, mult mai eficient şi adaptat platformei hardware decât le-ar putea realiza el însuşi.

În redarea obiectelor tridimensionale, biblioteca OpenGL foloseşte un număr redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor şi scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicaţie, pe baza acestor primitive. Dat fiind că OpenGL prevede un set puternic, dar de nivel scăzut, de comenzi de redare a obiectelor, mai sunt folosite şi alte biblioteci de nivel mai înalt care utilizează aceste comenzi şi preiau o parte din sarcinile de programare grafică. Astfel de biblioteci sunt:

- Biblioteca de funcţii utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafeţelor curbe şi alte funcţii grafice.

- Pentru fiecare sistem Windows există o extensie a bibliotecii OpenGL care asigură interfaţa cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc sistemul X Window, extensia GLX, pentru sisteme IMB OS/2, extensia PGL.
- Biblioteca de dezvoltare GLUT (OpenGL Utility Toolkit) este un sistem de dezvoltare independent de sistemul Windows, care ascunde dificultăţile interfeţelor de aplicaţii Windows, punând la dispoziţie funcţii pentru crearea şi iniţializarea ferestrelor, funcţii pentru prelucrarea evenimentelor de intrare şi pentru execuţia programelor grafice bazate pe biblioteca OpenGL.

În orice aplicaţie OpenGL trebuie să fie incluse fişierele header `gl.h` şi `glu.h` sau `glut.h` (dacă este utilizată biblioteca GLUT). Toate funcţiile bibliotecii OpenGL încep cu prefixul `gl`, funcţiile GLU încep cu prefixul `glu`, iar funcţiile GLUT încep cu prefixul `glut`.

6.1 DEZVOLTAREA PROGRAMELOR GRAFICE FOLOSIND UTILITARUL GLUT

Biblioteca grafică OpenGL conţine funcţii de redare a primitivelor geometrice independente de sistemul de operare, de orice sistem Windows sau de platforma hardware. Ea nu conţine nici o funcţie pentru a crea sau administra ferestre de afişare pe display (windows) şi nici funcţii de citire a evenimentelor de intrare (mouse sau tastatură).

Pentru crearea şi administarea ferestrelor de afişare şi a evenimentelor de intrare se pot aborda mai multe soluţii: utilizarea directă a interfeţei Windows (Win32 API), folosirea compilatoarelor sub Windows, care conţin funcţii de acces la ferestre şi evenimente sau folosirea altor instrumente (utilitare) care înglobează interfaţa OpenGL în sistemul de operare respectiv.

Un astfel de utilitar este GLUT, care se compilează şi instalează pentru fiecare tip de sistem Windows. Header-ul glut.h trebuie să fie inclus în fişierele aplicaţiei, iar biblioteca glut.lib trebuie legată (linkată) cu programul de aplicaţie. În lucrarea de faţă s-a folosit versiunea glut3.6, care poate fi preluată din Internet (www.sgi.com/pub/opengl/GLUT).

Sub GLUT, orice aplicaţie se structurează folosind mai multe funcţii callback. O funcţie callback este o funcţie care aparţine programului aplicaţiei şi este apelată de un alt program, în acest caz sistemul de operare, la apariţia anumitor evenimente. În GLUT sunt predefinite câteva tipuri de funcţii callback pentru iniţializarea programului, redimensionarea ferestrei de afişare, desenarea ferestrei şi controlul dispozitivelor de intrare (tastatură şi mouse). Aceste funcţii sunt scrise în aplicaţie şi pointerii lor sunt transmişi la înregistrare sistemului Windows, care le apelează (prin pointerul primit) în momentele necesare ale execuţiei.

6.1.1 FUNCȚII DE CONTROL AL FERESTREI DE AFIȘARE

Sunt disponibile cinci funcții pentru controlul ferestrei de afişare a programului. Aici trebuie să fie precizate diferitele semnificații ale termenului suprautilizat "fereastră". În general, în programare se foloseşte termenul fereastră (*window*) pentru orice zonă de afişare controlată pe display; ferestrele sunt gestionate de sistemul de operare (în cazul sistemului Microsoft Windows (95, NT) sau de subsistemul grafic X Window, care se execută sub sistemele de operare de tip Unix.

În aplicațiile de grafică tridimensională termenul de fereastră de vizualizare (*view plane window*) se referă la regiunea rectangulară care reprezintă intersecția dintre planul de vizualizare şi volumul de vizualizare şi în care sunt proiectate toate obiectele vizibile ale scenei. Fereastra de vizualizare se afişează într-o poartă de afişare (*viewport*), care se defineşte prin transformarea fereastră-poartă. Poarta de afişare se amplasează într-o fereastră de afişare (*window*) şi legătura dintre acestea este realizată diferit, în funcție de modul de programare folosit (interfață Windows sau toolkit).

În programele dezvoltate sub GLUT, pentru corelarea dintre poarta de afişare şi fereastra de afişare se folosesc funcțiile `glutInit()` şi `glutInitDisplayMode()`. Funcția:

```
void glutInit(int* argc, char** argv);
```

inițializează biblioteca GLUT folosind argumentele din linia de comandă; ea trebuie să fie apelată înaintea oricăror alte funcții GLUT sau OpenGL. Funcția:

```
void glutInitDisplayMode(unsigned int mode);
```

specifică caracteristicile de afişare a culorilor şi a bufferului de adâncime şi numărul de buffere de imagine. Parametrul `mode` se obține prin SAU logic între valorile fiecărei opțiuni. De exemplu:

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
```

iniția'lizează afişarea culorilor în modul RGB, cu două buffere de imagine şi buffer de adâncime. Alte valori ale parametrului `mode` sunt: `GLUT_SINGLE` (un singur buffer de imagine), `GLUT_RGBA` (modelul RGBA al culorii), `GLUT_STENCIL` (validare buffer şablon), `GLUT_ACCUM` (validare buffer de acumulare).

Funcția:

```
void glutInitWindowPosition(int x, int y);
```

specifică poziția pe ecran a colțului stânga sus al ferestrei de afişare. Pentru definirea dimensiunii inițiale a ferestrei de afişare se apelează funcția:

```
void glutInitWindowSize(int width, int heigth);
```

Crearea fereastrei în care se afişează contextul de redare (poarta) OpenGL are loc la apelul funcției:

```
int glutCreateWindow(char* string);
```

6.1.2 FUNCȚII CALLBACK

Funcțiile callback se definesc în program şi se înregistrează în sistem prin intermediul unor funcții GLUT. Ele sunt apelate de sistemul de operare atunci când este necesar, în funcție de evenimentele apărute. Funcția:

```
glutDisplayFunc(void(*Display)(void));
```

înregistrează funcția callback `Display()` care calculează şi afişează imaginea. Argumentul funcției este un pointer la o funcție fără argumente care nu returnează nici o valoare. Funcția `Display` (a aplicației) este apelată oridecâte ori este necesară desenarea ferestrei: la inițializare, la modificarea dimensiunilor ferestrei, sau la apelul explicit al funcției `gluPostRedisplay()`. Funcția:

```
glutReshapeFunc(void(*Reshape)(int w, int h));
```

înregistrează funcția callback `Reshape()` care este apelată oridecâte ori se modifică dimensiunea ferestrei de afişare. Argumentul este un pointer la funcția cu numele `Reshape` cu două argumente de tip întreg şi care nu returnează nici o valoare. În această funcție, programul de aplicație trebuie să refacă transformarea fereastră-poartă, dat fiind că fereastra de afişare şi-a modificat dimensiunile.

Funcția:

```
glutKeyboardFunc(void(*Keyboard)(unsigned int key,
                 int x, int y);
```

înregistrează funcția callback `Keyboard()` care este apelată atunci când se acționează o tastă. Parametrul `key` este codul tastei, iar `x` şi `y` sunt coordonatele (relativ la fereastra de afişare) a mouse-ului în momentul acționării tastei.

Funcția:

```
glutMouseFunc(void(*MouseFunc)(unsigned int button,
              int state, int x, int y);
```

înregistrează funcția callback `MouseFunc` care este apelată atunci când este apăsat sau eliberat un buton al mouse-ului. Parametrul `button` este codul butonului (poate avea una din constantele `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` sau `GLUT_RIGHT_BUTTON`). Parametrul `state` indică apăsarea (`GLUT_DOWN`) sau eliberarea (`GLUT_UP`) al unui buton al mouse-ului. Parametrii x şi y sunt coordonatele relativ la fereastra de afişare a mouse-ului în momentul evenimentului.

Execuția unui program folosind toolkit-ul GLUT se lansează prin apelul funcției `glutMainLoop()` după ce au fost efectuate toate inițializările şi înregistrările funcțiilor callback. Această buclă de execuție poate fi oprită prin închiderea ferestrei aplicației. În cursul execuției sunt apelate funcțiile callback în momentele apariției diferitelor evenimente. În cursul execuției se mai poate executa un proces atunci cînd nu apare nici un eveniment (atunci când coada de evenimente este vidă). Acest proces, folosit în animație, execută funcția callback `IdleFunc()` înregistrată prin apelul funcției:

```
glutIdleFunc(void(*IdleFunc)(void));
```

6.1.3 GENERAREA OBIECTELOR TRIDIMENSIONALE

Multe programe folosesc modele simple de obiecte tridimensionale pentru a ilustra diferite aspecte ale prelucrărilor grafice. GLUT conține câteva funcții care permit redarea unor astfel de obiecte tridimensionale în modul wireframe sau cu suprafețe pline (*filled*). Fiecare obiect este reprezentat într-un sistem de referință local, iar dimensiunea lui poate fi transmisă ca argument al funcției respective. Poziționarea şi orientarea obiectelor în scenă se face în programul de aplicație. Exemple de astfel de funcții sunt:

```
void glutWireCube(GLdouble size);
void glutSolidCube(GLdouble size);
void glutWireSphere(GLdouble radius, GLint slices,
                    GLint stacks);
void glutSolidSphere(GLdouble radius, GLint slices,
                    GLint stacks);
```

Programele GLUT au un mod specific de organizare, care provine din felul în care sunt definite şi apelate funcții callback. Acest mod va fi prezentat la primul exemplu de program OpenGL-GLUT şi va fi reluat apoi şi în alte exemple.

6.2 CARACTERISTICILE BIBLIOTECII OPENGL

În fişierul header ale bibliotecii OpenGL (gl.h) sunt definite mai multe constante simbolice, care reprezintă diferite stări, variabile sau valori de selecție a opțiunilor OpenGL. Aceste constante sunt toate scrise cu majuscule şi sunt precedate de prefixul `GL_`. De exemplu, constantele simbolice care definesc valorile `TRUE` şi `FALSE` şi cele care selectează tipul unei primitive grafice sunt:

```
/* Boolean */
#define GL_TRUE                         1
#define GL_FALSE                        0

/* BeginMode */
#define GL_POINTS                       0x0000
#define GL_LINES                        0x0001
#define GL_LINE_LOOP                    0x0002
#define GL_LINE_STRIP                   0x0003
#define GL_TRIANGLES                    0x0004
#define GL_TRIANGLE_STRIP               0x0005
#define GL_QUADS                        0x0007
#define GL_POLYGON                      0x0009
```

De asemenea, biblioteca OpenGL defineşte propriile tipuri de date, cele mai multe corespunzând tipurilor de date fundamentale ale limbajului C. De exemplu, în gl.h sunt definite următoarele tipuri:

```
typedef unsigned int GLenum;
typedef unsigned char GLboolean;
typedef signed char GLbyte;
typedef unsigned char GLubyte;
typedef short GLshort;
typedef int GLint;
typedef int GLsizei;
typedef unsigned int GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
typedef double GLclampd;
typedef void GLvoid;
```

Pentru a înțelege funcționarea comenzilor OpenGL, se descriu în continuare cele mai importante dintre caracteristicile OpenGL.

6.2.1 POARTA DE AFIŞARE OPENGL

Poarta de afişare OpenGL mai este numită *context de redare* (*rendering context*) şi este asociată unei ferestre de afişare din sistemul Windows. Dacă se programează folosind biblioteca GLUT, corelarea dintre fereastra de afişare şi poarta OpenGL este asigurată de funcții ale acestei biblioteci. Dacă nu se foloseşte biblioteca GLUT, atunci funcțiile bibliotecilor de extensie XGL, WGL sau PGL permit asocierea contextului de redare OpenGL cu o fereastră de afişare şi accesul la aceasta. Funcția OpenGL care defineşte transformarea fereastră-poartă este:

```
void glViewport(GLint x, GLint y,
                GLsizei width, GLsizei height);
```

unde x şi y specifică poziția colțului stânga-jos al dreptunghiului porții în fereastra de afişare (window) şi au valorile implicite 0, 0. Parametrii `weidth` şi `heigth` specifică lățimea, respectiv înălțimea, porții de afişare, dată ca număr de pixeli.

În OpenGL un pixel este reprezentat printr-un descriptor care defineşte mai mulți parametri:

- numărul de biți/pixel pentru memorarea culorii
- numărul de biți/pixel pentru memorarea adâncimii
- numărul de buffere de imagine.

6.2.2 BUFFERUL DE CADRU

Bufferul de cadru (*frame buffer*) conține toate datele care definesc o imagine şi constă din mai multe secțiuni logice: bufferul de imagine (sau bufferul de culoare), bufferul de adâncime (*Z-buffer*), bufferul şablon (*stencil*), bufferul de acumulare (*accumulation*).

Bufferul de imagine (*image buffer, color buffer*) în OpenGL poate conține una sau mai multe secțiuni, în fiecare fiind memorată culoarea pixelilor din poarta de afişare. Redarea imaginilor folosind un singur buffer de imagine este folosită

pentru imagini statice, cel mai frecvent în proiectarea grafică (CAD). În generarea interactivă a imaginilor dinamice, un singur buffer de imagine produce efecte nedorite, care diminuează mult calitatea imagii generate.

Orice cadru de imagine începe cu ştergerea (de fapt, umplerea cu o culoare de fond) a bufferului de imagine. După aceasta sunt generați pe rând pixelii care aparțin tuturor elementelor imaginii (linii, puncte, suprafețe) şi intensitățile de culoare ale acestora sunt înscrise în bufferul de imagine. Pentru trecerea la cadrul următor, trebuie din nou şters bufferul de imagine şi reluată generarea elementelor componente, pentru noua imagine. Chiar dacă ar fi posibilă generarea şi înscrierea în buffer a elementelor imaginii cu o viteză foarte mare (ceea ce este greu de realizat), tot ar exista un interval de timp în care bufferul este şters şi acest lucru este perceput ca o pâlpâire a imaginii. În grafica interactivă timpul necesar pentru înscrierea datelor în buffer este (în cazul cel mai fericit) foarte apropiat de intervalul de schimbare a unui cadru a imaginii (update rate) şi, dacă acest proces are loc simultan cu extragerea datelor din buffer şi afişarea lor pe display, atunci ecranul va prezenta un timp foarte scurt imaginea completă a fiecărui cadru, iar cea mai mare parte din timp ecranul va fi şters sau va conține imagini parțiale ale cadrului. Tehnica universal folosită pentru redarea imaginilor dinamice (care se schimbă de la un cadru la altul) este tehnica *dublului buffer de imagine*.

În această tehnică există două buffere de imagine: bufferul din față (*front*), din care este afişată imaginea pe ecran şi bufferul din spate (*back*), în care se înscriu elementele de imagine generate. Când imaginea unui cadru a fost complet generată, ea poate fi afişată pe ecran printr-o simplă operație de comutare între buffere: bufferul spate devine buffer față, şi din el urmează să fie afişată imagine cadrului curent, iar bufferul față devine buffer spate şi în el urmează să fie generată imaginea noului cadru. Comutarea între bufferele de imagine se poate sincroniza cu cursa de revenire pe verticală a monitorului şi atunci imaginile sunt prezentate continuu, fără să se observe imagini fragmentate sau pâlpâiri.

În OpenGL comutarea bufferelor este efectuată de funcția `SwapBuffers()`. Această funcție trebuie apelată la terminarea generării imaginii tuturor obiectelor vizibile în fiecare cadru.

Modul în care se execută comutarea bufferelor depinde de platforma hardware. În lipsa unui accelerator grafic, toate operațiile OpenGL sunt executate software, bufferul spate este o zonă din memorie principală, iar bufferul față este memoria video din adaptorul grafic. La comutarea bufferelor care se produce la apelul funcției `SwapBuffers()` are loc copierea bufferului back (care este implementat ca un bitmap independent de dispozitiv - DIB - *device independent bitmap*), în bufferul front, care este memoria de ecran a sistemului grafic.

Biblioteca OpenGL oferă posibilitatea creerii imaginilor cu simplu sau dublu buffer, monoscopice şi stereoscopice.

Funcția `void glDrawBuffer(GLenum mode)` stabileşte bufferul în care se desenează primitivele geometrice. Parametrul `mode` poate lua una din valorile: `GL_LEFT`, `GL_RIGHT`, `GL_FRONT`, `GL_BACK`, `GL_FRONT_LEFT`, `GL_FRONT_RIGHT`, `GL_BACK_LEFT`, `GL_BACK_RIGHT`, `GL_NONE`.

Valoarea implicită este `GL_FRONT` pentru imagini cu un singur buffer şi `GL_BACK` pentru imagini cu dublu buffer. Celelate opţiuni se referă la imagini stereoscopice.

Funcţia `glReadBuffer(GLenum mode)` stabileşte bufferul curent din care se citesc pixelii sursă în operaţii de combinare sau acumulare. Parametrul `mode` poate lua una din valorile descrise anterior. Valoarea implicită este `GL_FRONT` pentru imagini monoscopice cu un singur buffer de culoare sau `GL_BACK` pentru imagini monoscopice cu două buffere de culoare.

Pentru imaginile stereoscopice, biblioteca OpenGL pune la dispoziţie bufferele necesare pentru definirea celor două imagini diferite (stânga, dreapta), pentru simplu şi dublu buffer. Crearea propriu-zisă a imaginilor stereoscopice este rezolvată la nivelul programului de aplicaţie şi necesită dispozitive de afişare adecvate.

Bufferul de adâncime (*depth buffer*) memorează adâncimea fiecărui pixel şi, prin aceasta, permite eliminarea suprafeţelor ascunse. Bufferul de adâncime conţine acelaşi număr de locaţii ca şi un buffer de imagine, fiecare locaţie corespunzând unui pixel, de o anumită adresă. Valoarea memorată în locaţia corespunzătoare unui pixel este distanţa faţă de punctul de observare (adâncimea pixelului). La generarea unui nou pixel cu aceeaşi adresă, se compară adâncimea noului pixel cu adâncimea memorată în bufferul de adâncime, şi noul pixel înlocuieşte vechiul pixel (îl "ascunde") dacă este mai apropiat de punctul de observare. Bufferul de adâncime se mai numeşte şi Z-buffer, de la coordonata z, care reprezintă adâncimea în sistemul de referinţă ecran 3D.

Bufferul şablon (*stencil buffer*) este un buffer auxiliar utilizat pentru restricţionarea desenării în anumite zone ale porţii de afişare. Se pot obţine astfel imagini încadrate în anumite şabloane, ca de exemplu, instrumente de afişare folosite în simulatoare de antrenament. Bufferul şablon poate fi folosit şi pentru redarea suprafeţelor coplanare.

Bufferul de acumulare (*accumulation buffer*) este folosit pentru crearea imaginilor anti-alising prin acumularea intensităţilor de culoare a pixelilor rezultaţi prin eşantionarea imaginii.

6.2.3 OPERAŢIILE DE BAZĂ OPENGL

OpenGL desenează *primitive geometrice* (puncte, linii şi poligoane) în diferite moduri selectabile. Primitivele sunt definite printr-un grup de unul sau mai multe vârfuri (*vertices*). Un vârf defineşte un punct, capătul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date:

- coordonate
- culoare
- normală
- coordonate de textură.

Aceste date sunt prelucrate independent, în ordine şi în acelaşi mod pentru toate primitivele geometrice. Singura deosebire care apare este aceea că, dacă o linie sau o suprafaţă este decupată, atunci grupul de vârfuri care descriau iniţial primitiva respectivă este înlocuit cu un alt grup, în care pot apare vârfuri noi rezultate din intersecţia laturilor cu planele volumului de decupare (volumul canonic) sau unele vârfuri din cele iniţiale pot să dispară.

Comenzile sunt prelucrate în ordinea în care au fost trimise şi orice comandă trebuie executată complet înainte ca o nouă comandă să fie executată. OpenGL este o interfaţă procedurală, secvenţa de comenzi (apeluri de funcţii) descriu ce operaţii să fie executate şi nu ce rezultat s-ar dori obţinut. Se pot specifica valori ale matricelor de transformare, tipul proiecţiei, coeficienţi de reflexie, etc.

Modelul de execuţie OpenGL este modelul *client-server*. O aplicaţie (*client*) generează comenzi care sunt interpretate şi executate de către OpenGL (*server*). Serverul OpenGL poate să fie executat pe un alt calculator decât aplicaţia, dar atunci trebuie definit un protocol de comunicaţie client-server, dat fiind că nu există comenzi OpenGL pentru astfel de operaţii.

Efectul comenzilor OpenGL asupra bufferului de imagine este controlat de sistemul Windows care alocă zona de afişare a porţii (viewport) OpenGL pe ecran.

Fig. 6.1 prezintă o diagramă bloc simplificată a modului în care OpenGL prelucrează datele. S-au ignorat pentru început aspectele legate de texturare şi anti-aliasing. Secvenţa de operaţii OpenGL (numită *pipeline grafic*) se execută asupra vârfurilor şi asupra datelor asociate acestora: coordonate, culoare, normală (şi coordonate de textură, care nu sunt reprezentate în figură).

Biblioteca OpenGL primeşte o succesiune de primitive geometrice pe care le desenează, adică le converteşte în mulţimea de pixeli corespunzătoare, înscriind valorile culorilor acestora într-un buffer de imagine. În continuare sunt detaliate fiecare dintre operaţiile grafice prezentate în figură.

Modul în care este executată secvenţa de operaţii pentru redarea primitivelor grafice depinde de starea bibliotecii OpenGL, stare care este definită prin mai multe variabile de stare ale acesteia (parametri). Numărul de variabile de stare ale bibliotecii este destul de mare, descrierea lor poate fi găsită în manualul de referinţă (*OpenGL Reference Manual*), iar pe parcursul expunerii vor fi prezentate numai cele mai importante dintre acestea.

La iniţializare, fiecare variabilă de stare este setată la o valoare implicită. O stare o dată setată îşi menţine valoarea neschimbată până la o nouă setare. Variabilele de stare au denumiri date sub formă de constante simbolice care pot fi folosite pentru aflarea valorilor acestora. Câteva exemple de stări definite prin constante simbolice în fişierul gl.h sunt:

```
#define GL_CURRENT_COLOR                 0x0B00
#define GL_CURRENT_NORMAL                0x0B02
#define GL_POLYGON_MODE                  0x0B40
#define GL_FOG_COLOR                     0x0B66
#define GL_MODELVIEW_MATRIX              0x0BA6
#define GL_PROJECTION_MATRIX             0x0BA7
```

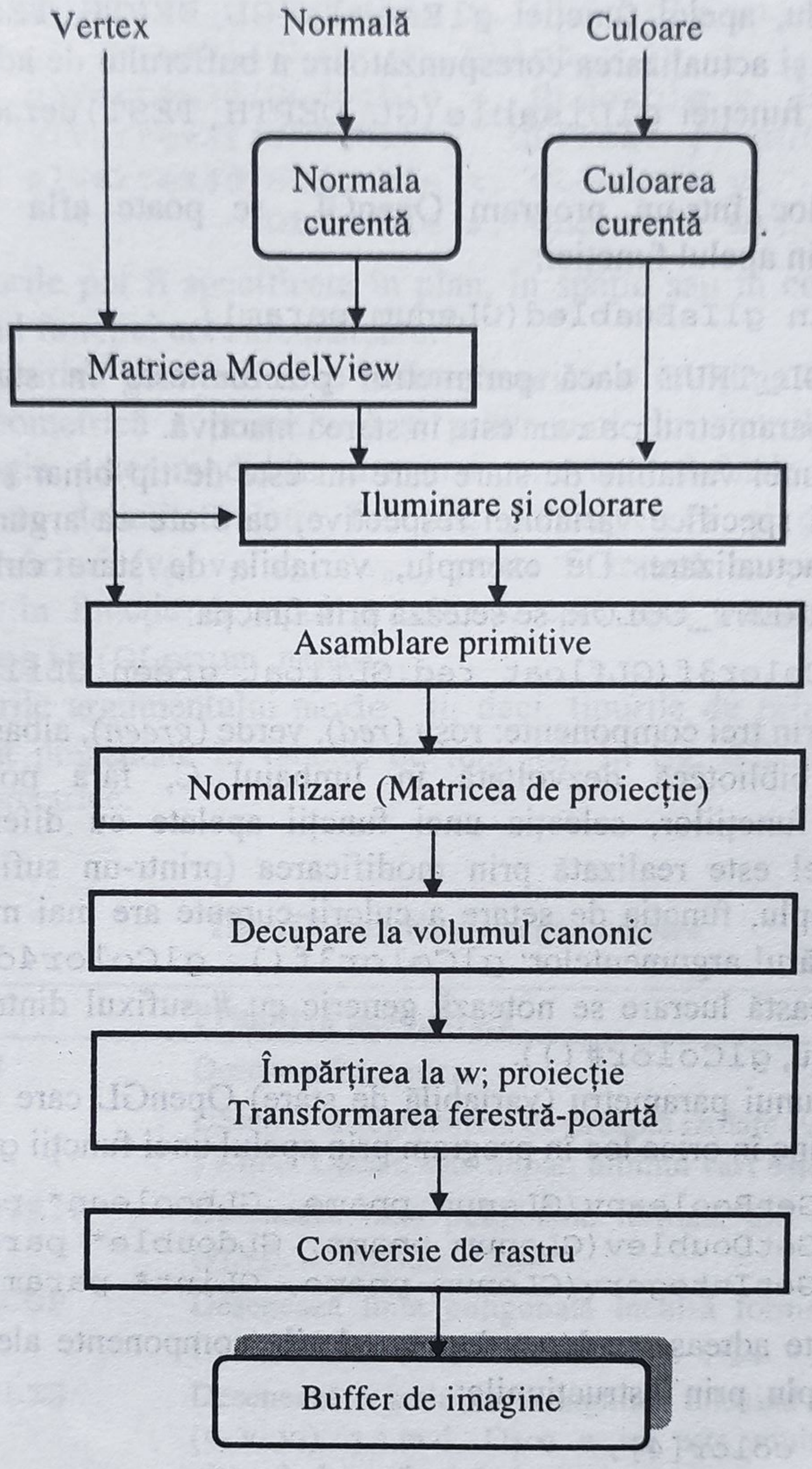


Fig. 6.1 Operaţiile grafice OpenGL

Variabilele de stare OpenGL sunt de două categorii: variabile de tip binar şi variabile definite prin diferite structuri de date.

Variabile de tip binar pot avea una din două stări: starea activă (*enabled*) sau starea inactivă (*disabled*). Setarea la starea activă se realizează prin apelul funcţiei:

```
void glEnable(GLenum param);
```

unde `param` este numele simbolic al parametrului (variabilei de stare).

Setarea la starea inactivă se realizează prin apelul funcţiei:

```
void glDisable(GLenum param);
```

De exemplu, apelul funcției `glEnable(GL_DEPTH_TEST)` activează testul de adâncime şi actualizarea corespunzătoare a bufferului de adâncime (*depth buffer*), iar apelul funcției `glDisable(GL_DEPTH_TEST)` dezactivează testul de adâncime.

În orice loc într-un program OpenGL, se poate afla valoarea unui parametru binar prin apelul funcției:

```
GLboolean glIsEnabled(GLenum param);
```

care returnează `GL_TRUE` dacă parametrul `param` este în starea activă şi `GL_FALSE` dacă parametrul `param` este în starea inactivă.

Valoarea unei variabile de stare care nu este de tip binar se setează prin apelul unei funcții specifice variabilei respective, care are ca argumente valorile necesare pentru actualizare. De exemplu, variabila de stare culoare curentă, denumită `GL_CURRENT_COLOR`, se setează prin funcția:

```
void glColor3f(GLfloat red,GLfloat green,GLfloat blue);
```

la o valoare dată prin trei componente: roşu (*red*), verde (*green*), albastru (*blue*).

Fiind o bibliotecă dezvoltată în limbajul C, fără posibilitatea de supraîncărcare a funcțiilor, selecția unei funcții apelate cu diferite tipuri de argumente de apel este realizată prin modificarea (printr-un sufix) a numelui funcției. De exemplu, funcția de setare a culorii curente are mai multe variante, după tipul şi numărul argumentelor: `glColor3f()`, `glColor4d()`, etc. În continuare, în această lucrare se notează generic cu # sufixul dintr-o familie de funcții (de exemplu, `glColor#()`).

Valoarea unui parametru (variabilă de stare) OpenGL care nu este de tip binar se poate obține în orice loc în program prin apelul unei funcții `glGet#()`:

```
void glGetBooleanv(GLenum pname, GLboolean* params);
void glGetDoublev(GLenum pname, GLdouble* params);
void glGetIntegerv(GLenum pname, GLint* params);
```

unde `params` este adreasa unde se depun valorile componente ale parametrului `pname`. De exemplu, prin instrucțiunile:

```
GLfloat color[4];
glGetFloatv(GL_CURRENT_COLOR, color);
```

se obțin cele patru componente ale culorii curente (red, green, blue, alpha) în vectorul `color`.

6.2.4 PRIMITIVE GEOMETRICE

Funcțiile OpenGL execută secvența de operații grafice asupra fiecărei primitive geometrice, definită printr-o mulțime de vârfuri şi tipul acesteia. Coordonatele unui vârf sunt transmise către OpenGL prin apelul unei funcții `glVertex#()`. Aceasta are mai multe variante, după numărul şi tipul argumentelor. Iată, de exemplu, numai câteva din prototipurile funcțiilor `glVertex#()`:

```
void glVertex2d(GLdouble x, GLdouble y);
void glVertex2i(GLint x, GLint y );
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
void glVertex4d(GLdouble x, GLdouble y,
                GLdouble z, GLdouble w);
```

Vârfurile pot fi specificate în plan, în spațiu sau în coordonate omogene, folosind apelul funcției corespunzătoare.

O primitivă geometrică se defineşte printr-o mulțime de vârfuri (care dau descrierea geometrică a primitivei) şi printr-unul din tipurile prestabilite, care indică topologia, adică modul în care sunt conectate vârfurile între ele. Mulțimea de vârfuri este delimitată între funcțiile `glBegin()` şi `glEnd()`. Aceeaşi mulțime de vârfuri (v_0, v_1, v_2,....v_{n-1}) poate fi tratată ca puncte izolate, linii, poligon, etc, în funcție de tipul primitivei, care este transmis ca argument al funcției `glBegin(GLenum mode)`.

Valorile argumentului `mode` şi, deci, tipurile de primitive acceptate de OpenGL sunt prezentate în tabelul de mai jos. În fig. 6.2 sunt ilustrate aceste primitive geometrice.

Tabelul 6.1

Tipurile de primitive geometrice

Argument	Primitivă geometrică
GL_POINTS	Desenează n puncte
GL_LINES	Desenează segmentele de dreaptă izolate (v_0,v_1), (v_2,v_3),... ş.a.m.d. Dacă n este impar, ultimul vârf este ignorat.
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele (v_0,v_1), (v_1,v_2), ...(v_{n-2}, v_{n-1}).
GL_LINE_LOOP	Desenează linia poligonală închisă formată din segmentele (v_0,v_1), (v_1,v_2), ...(v_{n-2}, v_{n-1}), (v_{n-1}, v_1) .
GL_TRIANGLES	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_3,v_4,v_5), ş.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
GL_TRIANGLE_ STRIP	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_2,v_1,v_3),..ş.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeaşi orientare.
GL_TRIANGLE_FAN	Desenează triunghiurile (v_0,v_1,v_2), (v_0,v_2,v_3), ş.a.m.d.
GL_QUADS	Desenează o serie de patrulatere (v_0,v_1,v_2,v_3), (v_4,v_5,v_6,v_7), ş.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
GL_QUADS_ STRIP	Desenează o serie de patrulatere (v_0,v_1,v_3,v_2), (v_3,v_2,v_5,v_4), ş.a.m.d. Dacă n < 4, nu se desenază nimic. Dacă n este impar, ultimul vârf este ignorat.
GL_POLYGON	Desenează un poligon cu n vârfuri, (v_0,v_1,...v_{n-1}). Dacă poligonul nu este convex, rezultatul este impredictibil.

Acest tabel, împreună cu fig. 6.2 sunt suficiente pentru înțelegerea modului în care OpenGL interpretează primitivele geometrice.

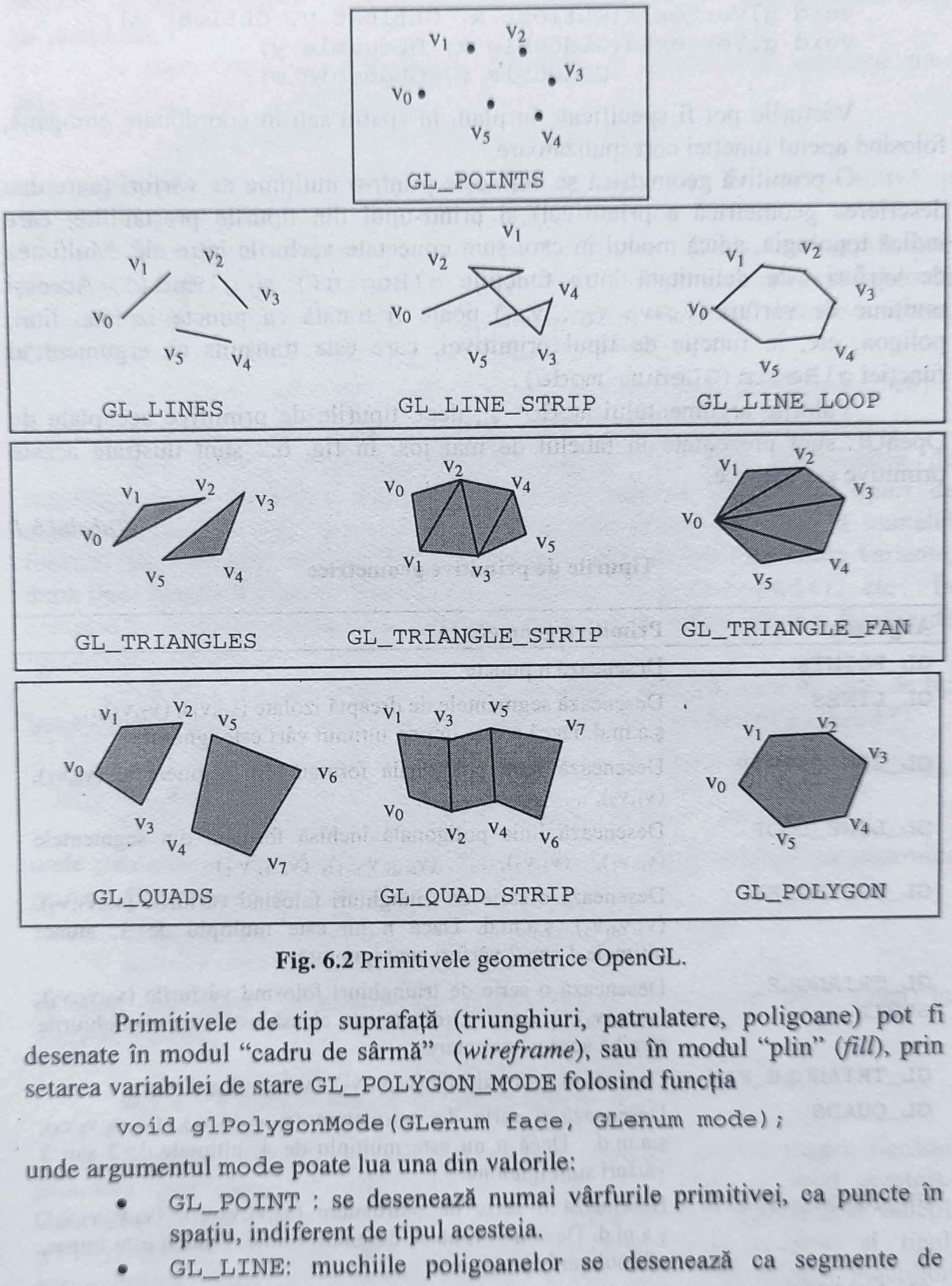


Fig. 6.2 Primitivele geometrice OpenGL.

Primitivele de tip suprafață (triunghiuri, patrulatere, poligoane) pot fi desenate în modul "cadru de sârmă" (*wireframe*), sau în modul "plin" (*fill*), prin setarea variabilei de stare `GL_POLYGON_MODE` folosind funcția

```
void glPolygonMode(GLenum face, GLenum mode);
```

unde argumentul `mode` poate lua una din valorile:

- `GL_POINT` : se desenează numai vârfurile primitivei, ca puncte în spațiu, indiferent de tipul acesteia.
- `GL_LINE`: muchiile poligoanelor se desenează ca segmente de dreaptă.
- `GL_FILL`: se desenează poligonul plin.

Argumentul `face` se referă la tipul primitivei geometrice (din punct de vedere al orientării), căreia i se aplică modul de redare `mode`. Din punct de vedere al orientării, OpenGL admite primitive orientate direct şi primitive orientate invers. Argumentul `face` poate lua una din valorile: `GL_FRONT`, `GL_BACK` sau `GL_FRONT_AND_BACK`, pentru a se specifica primitive orientate direct, primitive orientate invers şi, respectiv, ambele tipuri de primitive.

În mod implicit, sunt considerate orientate direct suprafeţele ale căror vârfuri sunt parcurse în ordinea inversă acelor de ceas. Acestă setare se poate modifica prin funcţia `glFrontFace(GLenum mode)` unde `mode` poate lua valoarea `GL_CCW` pentru orientare în sens invers acelor de ceas (*counterclockwise*) sau `GL_CW` pentru orientare în sensul acelor de ceasornic (*clockwise*).

6.2.5 REPREZENTAREA CULORILOR ÎN OPENGL

În biblioteca OpenGL sunt definite două modele de culori: modelul de culori RGBA şi modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B şi transparenţa A pentru fiecare primitivă geometrică sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentată printr-un index într-o tabelă de culori (*color map*), care are memorate pentru fiecare intrare (index) componentele corespunzătoare R,G,B,A ale culorii. În modul de culori indexate numărul de culori afişabile simultan este limitat de dimensiunea tabelei culorilor şi, în plus, nu se pot efectua unele dintre prelucrările grafice importante (cum sunt umbrirea, anti-aliasing, ceaţa). Modelul de culori indexate este folosit în principal în aplicaţii de proiectare grafică (CAD), în care este necesar un număr mic de culori şi nu se folosesc umbrirea, ceaţa, etc. În aplicaţiile de realitate virtuală nu se poate folosi modelul de culori indexate şi de aceea în continuare nu vor mai fi prezentate comenzile sau opţiunile care se referă la acest model şi toate descrierile consideră numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitivă geometrică depinde de mai multe condiţii, putând fi o culoare constantă a primitivei, o culoare calculată prin interpolare între culorile vârfurilor primitivei, sau o culoare calculată în funcţie de iluminare, anti-aliasing şi texturare. Presupunând pentru moment culoarea constantă a unei primitive, aceasta se obţine prin setarea unei variabile de stare a bibliotecii, variabila de culoare curentă (`GL_CURRENT_COLOR`). Culoarea curentă se setează folosind una din funcţiile `glColor#()`, care are mai multe variante, în funcţie de tipul şi numărul argumentelor. De exemplu, câteva din prototipurile acestei funcţii definite în fişierul gl.h sunt:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
void glColor3ub(GLubyte r, GLubyte g, GLubyte b);
void glColor4d(GLdouble r, GLdouble g,
               GLdouble b, GLdouble a);
```

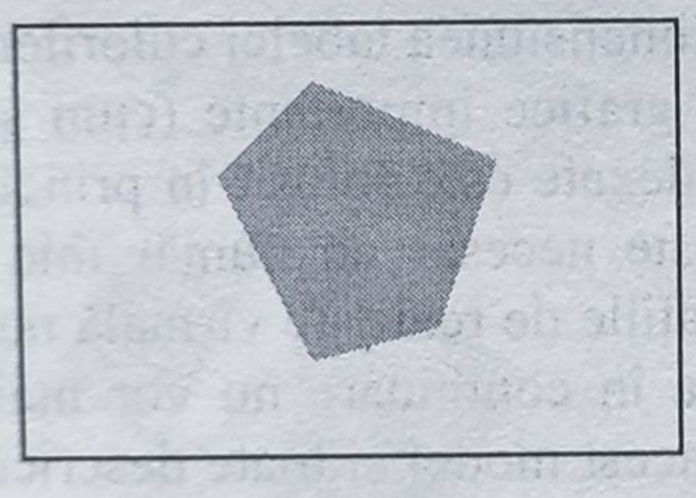
Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor roşu (r), verde (g), albastru (b), respectiv transparenţă (a) ca a patra

componentă pentru funcţiile cu 4 argumente. Fiecare componentă a culorii curente este memorată ca un număr în virgulă flotantă cuprins în intervalul [0,1]. Valorile argumentelor de tip întreg fără semn (unsigned int, unsigned char, etc.) sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea 0 este transformată în 0.0, iar valoarea maximă reprezentabilă este transformată în 1.0 (intensitate maximă). Valorile argumentelor de tip întreg cu semn sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea negativă maximă este transformată în –1.0, iar valoarea pozitivă maximă este transformată în 1.0 (intensitate maximă).

Valoarea finală a culorii unui pixel, rezultată din toate calculele de umbrire, anti-aliasing, texturare, etc, este memorată în bufferul de imagine, o componentă fiind reprezentată printr-un număr `n` de biţi (nu neapărat acelaşi număr de biţi pentru fiecare componentă). Deci componentele culorilor pixelilor memorate în bufferul de imagine sunt numere întregi în intervalul $(0 , 2^n - 1)$, care se obţin prin conversia numerelor în virgulă flotantă prin care sunt reprezentate şi prelucrate culorile primitivelor geometrice, ale materialelor, etc.

■ Exemplul 6.1

Secvenţa de funcţii pentru redarea unui poligon în spaţiul tridimensional şi imaginea corespunzătoare generată de OpenGL arată în felul următor:



```
glColor3f(0.8,0.8,0.8);
glBegin(GL_POLYGON);
    glVertex3d(-4,2,-20);
    glVertex3d(-1,5,-20);
    glVertex3d(5,2.5,-20);
    glVertex3d(3,-3,-20);
    glVertex3d(-1,-4,-20);
glEnd();
```

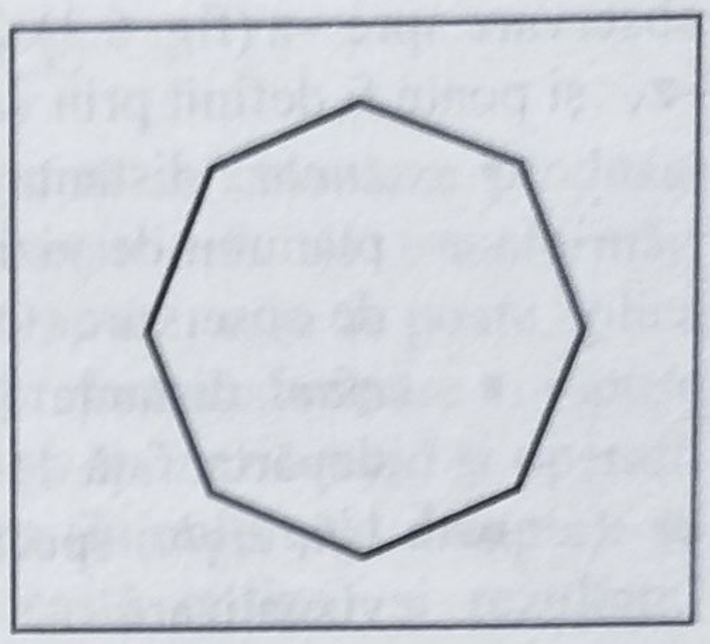
Indentarea instrucţiunilor cuprinse între `glBegin()` şi `glEnd()` nu este necesară în limbajul C în care este scris codul, ea este folosită doar pentru a evidenţia începutul şi sfârşitul primitivei geometrice.

În mod implicit, calculul transparenţei nu este validat. Apelul funcţiei `glEnable(GL_BLEND)` validează combinarea culorilor, prin care culoarea fiecărui pixel al poligonului este combinată cu culoarea pixelului existent în bufferul de imagine. Modul în care are loc combinarea se selectează prin funcţia `glBlendFunc()`. Prin combinarea culorilor se simulează transparenţa suprafeţelor precum şi alte efecte de redare a imaginilor (anti-aliasing, ceaţă). Exemple de transparenţă prin combinarea culorilor sunt date în §7.4.5.1.

■ Exemplul 6.2

Se poate genera un poligon regulat prin calculul coordonatelor vârfurilor şi folosirea lor ca argumente ale funcţiilor `glVertex#()`. De exemplu, calculul şi desenarea în OpenGL a conturului unui octogon în spaţiu este următoarea:

```
#define PI 3.141592
int n = 8;
double radius = 10;
glColor3d(0,0,0);
glBegin(GL_LINE_LOOP);
   for (int i=0;i<n;i++){
      double angle = 2*PI*i/n;
      glVertex3d(radius*cos(angle),
         radius*sin(angle),-20);
   }
glEnd();
```



6.3 SISTEMUL DE VIZUALIZARE OPENGL

Sistemul de vizualizare OpenGL defineşte sistemele de referinţă, transformările geometrice şi relaţiile (matriceale) de transformare pentru redarea primitivelor geometrice. Sistemul de vizualizare OpenGL este o particularizare a sistemului PHIGS, în care centrul sistemului de referinţă de observare (VRP) coincide cu centrul de proiecţie (PRP).

6.3.1 SISTEMELE DE REFERINŢĂ

Sistemul de referinţă universal definit în OpenGL este un sistem drept, iar matricea care reprezintă un punct (în plan, în spaţiul tridimensional sau în coordonate omogene) este matrice coloană.

Pentru reprezentarea unei matrice 4x4 folosite pentru transformări geometrice în coordonate omogene, în OpenGL este utilizată convenţia coloană majoră: fiind dat un număr de 16 valori reale (tip `float` sau `double`), în ordinea $a_0, a_1, a_2, \ldots a_{15}$, matricea corespunzătoare este:

$$A = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Aceste convenţii sunt aceleaşi cu convenţiile adoptate în lucrarea de faţă, deci matricele de transformări geometrice ale bibliotecii OpenGL sunt identice cu cele prezentate în secţiunile precedente.

Sistemul de referinţă de observare este un sistem de referinţă drept, definit ca localizare şi orientare relativ la sistemul de referinţă universal. În acest sistem de referinţă de observare se poate defini o transformare de proiecţie paralelă ortografică sau de proiecţie perspectivă.

Funcţiile oferite de OpenGL permit definirea proiecţiei perspective pe un plan perpendicular pe axa z a sistemului de referinţă de observare, cu direcţia de

observare spre –z (fig. 6.3). Trunchiul de piramidă de vizualizare este orientat spre $-z_V$ şi poate fi definit prin valorile:

- *znear* : distanţa (dată ca valoare pozitivă) a planului de vizualizare şi a planului de vizibilitate apropiat faţă de centrul sistemului de referinţă de observare (O_V)
- *zfar*: distanţa (dată ca valoare pozitivă) a planului de vizibilitate depărtat faţă de centrul sistemului de referinţă de observare
- *left, right*: specifică coordonatele planelor verticale ale volumului de vizualizare
- *top, bottom*: specifică coordonatele planelor orizontale ale volumului de vizualizare.

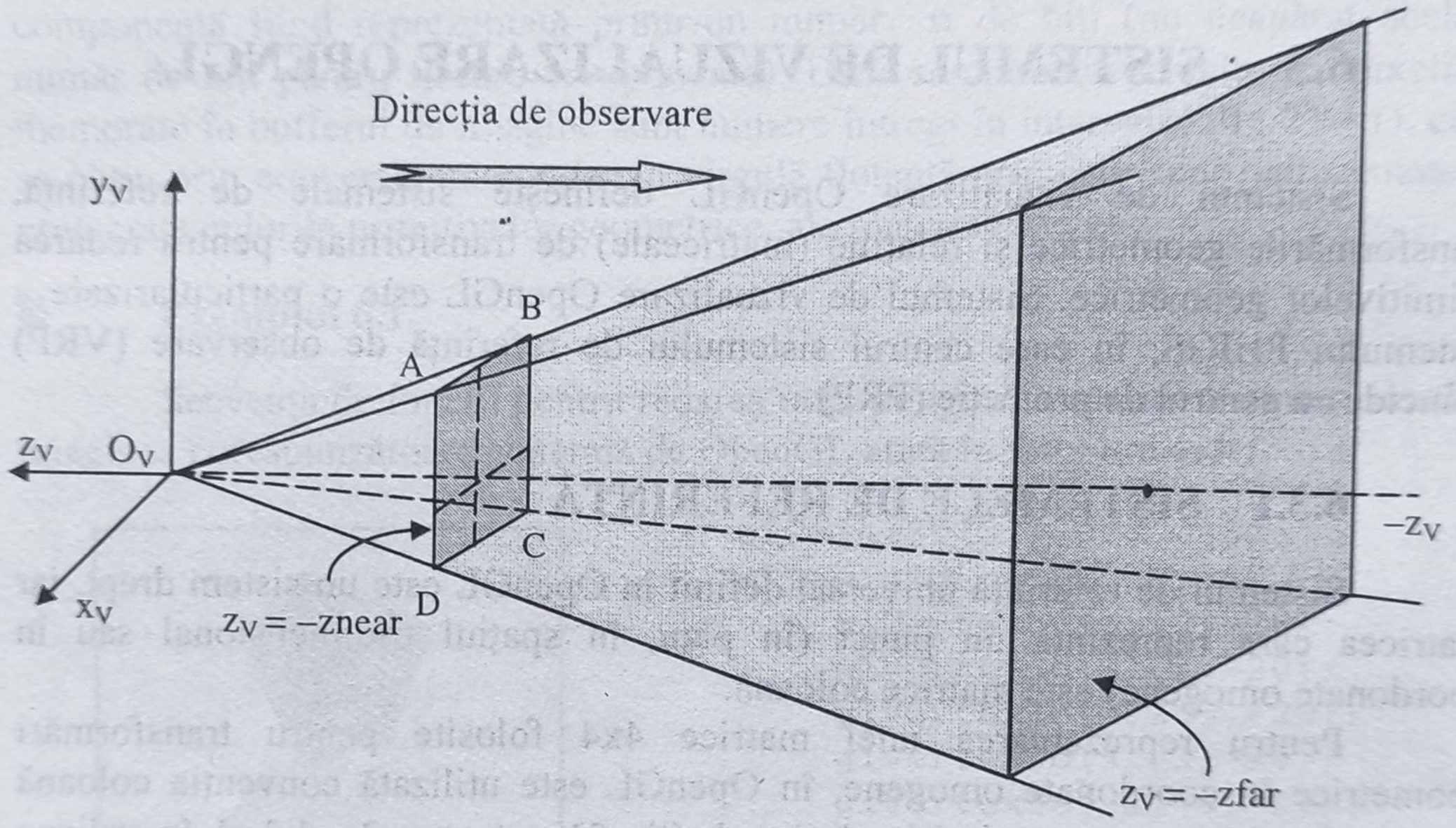


Fig. 6.3 Definirea proiecţiei perspectivă şi a volumului de vizualizare în OpenGL.

Colţul din stânga-jos (D) al ferestrei de vizualizare are coordonatele (`left, bottom, -znear`) în sistemul de referinţă de observare, iar colţul din dreapta-sus (B) are coordonatele (`right, top, -znear`) în acelaşi sistem.

Funcţia OpenGL care defineşte o astfel de proiecţie perspectivă este funcţia `glFrustum()` care are prototipul:

```
void glFrustum(GLdouble left, GLdouble right,
               GLdouble bottom, GLdouble top,
               GLdouble znear, GLdouble zfar);
```

Matricea de normalizare calculată în acest caz este o particularizare a produsului matricelor date de relaţiile (4.14), (4.15), (4.16), în care: n = – znear; d = – znear; f = – zfar; x_{min} = left; x_{max} = right; y_{min} = bottom; y_{max} = top. Deducerea ei este propusă ca un exerciţiu pentru cititori.

6.3.2 TRANSFORMĂRI GEOMETRICE

Aşa după cum s-a mai arătat, nu este eficient să se calculeze produsul dintre matricea de reprezentare a fiecărui punct şi matricele de transformări succesive, ci se calculează o matrice de transformare compusă, care se poate aplica unuia sau mai multor obiecte. Pentru calculul matricelor de transformare compuse (prin produs de matrice) se foloseşte o *matrice de transformare curentă* şi operaţii de acumulare în matricea curentă prin postmultiplicare (înmulţire la dreapta): se înmulţeşte matricea curentă cu noua matrice (în această ordine) şi rezultatul înlocuieşte conţinutul matricei curente.

Pentru secvenţa de transformări în ordinea $\mathbf{M_1}$, $\mathbf{M_2}$...$\mathbf{M_n}$, aplicate obiectelor, se iniţializează matricea curentă **C** cu matricea identitate, după care se calculează matricea compusă prin postmultiplicare în ordine inversă celei în care se aplică matricele componente:

$$\mathbf{C = I}$$
$$\mathbf{C = CM_n = M_n}$$
$$\mathbf{C = CM_{n-1} = M_nM_{n-1}}$$
$$\mathbf{.....}$$
$$\mathbf{C = CM_2 = M_nM_{n-1}.....M_2}$$
$$\mathbf{C = CM_1 = M_nM_{n-1}.....M_2M_1}$$
$$\mathbf{C = M = M_nM_{n-1}.....M_2M_1}$$
$$\mathbf{P' = MP = M_nM_{n-1}.....M_2M_1P}$$

În această transformare, se aplică mai întâi matricea $\mathbf{M_1}$ punctului **P**; punctului transformat rezultat i se aplică transformarea $\mathbf{M_2}$, ş.a.m.d. Se poate reţine deci că matricea compusă **M** se scrie ca produs în ordinea inversă aplicării transformărilor componente şi tot în ordine inversă se acumulează în matricea curentă. Acest mod de calcul al matricelor compuse are suport în OpenGL prin mai multe variabile interne (matrice curente) şi funcţii prin care acestea pot fi calculate. Pentru început, se consideră o matrice curentă oarecare **C** stabilită în OpenGL printr-o comandă corespunzătoare (va fi prezentată ulterior).

Matricea curentă se poate iniţializa cu matricea identitate prin funcţia `glLoadIdentity()` sau cu o matrice oarecare, dată prin pointer la un vector de 16 valori consecutive, prin funcţia `glLoadMatrix#()`.

```
glLoadMatrixd(const GLdouble* m);
glLoadMatrixf(const GLfloat* m);
```

Valorile din vectorul `GLdouble* m` (respectiv `GLfloat* m`) sunt atribuite în ordinea coloană majoră matricei curente.

Conţinutul matricei curente poate fi modificat prin multiplicare (la dreapta) cu o altă matrice, dată printr-un vector de 16 valori de tip `double` sau `float` utilizând funcţia `glMultMatrix#()`.

```
glMultMatrixd(const GLdouble* m);
glMultMatrixf(const GLfloat* m);
```

Matricea curentă **C** este înlocuită cu produsul **C M**, unde **M** este matricea corespunzătoare vectorului dat prin pointerul `m`. Crearea unei matrice pentru o transformare elementară (translație, scalare, etc) și înmulțirea ei cu matricea curentă se poate face prin apelul unei singure funcții OpenGL.

Transformarea de translație cu valorile x, y, z se implementează prin apelul uneia din funcțiile `glTranslated()` sau `glTranslatef()`, după tipul argumentelor de apel:

```
glTranslated(GLdouble x, GLdouble y, GLdouble z);
glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glTranslate#()` creează o matrice de translație **T**(x, y, z), dată de relația (3.2), înmulțește la dreapta matricea curentă **C**, iar rezultatul înlocuiește conținutul matricei curente **C**, deci: **C** = **C T**(x, y, z).

Transformarea de scalare este efectuată de una din funcțiile `glScaled()` sau `glScalef()`.

```
glScaled(GLdouble x, GLdouble y, GLdouble z);
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glScale#()` creează o matrice de scalare **S**(x, y, z) și o înmulțește cu matricea curentă, rezultatul fiind depus în matricea curentă.

Transformarea de rotație se definește printr-o direcție de rotație dată prin vectorul de poziție x, y, z și un unghi `angle` (specificat în grade). Prototipurile funcțiilor de rotație sunt:

```
glRotated(GLdouble angle, GLdouble x,
          GLdouble y, GLdouble z);
glRotatef(GLfloat angle, GLfloat x,
          GLfloat y, GLfloat z);
```

Rotațiile în raport cu axele de coordonate sunt cazuri particulare ale funcțiilor `glRotate#()`. De exemplu, rotația cu unghiul `angle` în raport cu axa `x` se obține la apelul funcției `glRotated(angle,1,0,0)`.

Transformările compuse se efectuează prin acumularea produsului matricelor componente în matricea curentă. De exemplu, transformările de modelare din exemplul 3.2 au fost implementate în OpenGL după cum urmează.

Transformarea $\mathbf{M}_2 = \mathbf{T}(0,-8,0)\,\mathbf{S}(2,2,2)$:

```
glTranslated(0,-8,0);
glScaled(2,2,2);
```

Transformarea $\mathbf{M}_3 = \mathbf{T}(8,0,0)\mathbf{R}_z(\pi/4)\mathbf{S}(2,1,2)$:

```
glTranslated(8,0,0);
glRotated(45,0,0,1);
glScaled(2,1,2);
```

Transformarea $\mathbf{M}_4 = \mathbf{T}(-8,0,0)\mathbf{R}_Z(-\pi/4)\mathbf{S}(2,1,2)$:

```
glTranslated(-8,0,0);
glRotated(-45,0,0,1);
glScaled(2,1,2);
```

Stivele matricelor de transformare. După cum s-a mai arătat, transformarea unui obiect din sistemul de referinţă local (de modelare) în sistemul de referinţă normalizat este compusă din succesiunea transformărilor de modelare (instanţiere) $\mathbf{M}_I$, de observare $\mathbf{M}_V$ şi de normalizare $\mathbf{M}_N$:

$$\mathbf{M}=\mathbf{M}_N\mathbf{M}_V\mathbf{M}_I$$

Transformarea de instanţiere $\mathbf{M}_I$ este, în general, specifică fiecărui obiect, deci se calculează pentru fiecare obiect în parte.

Pentru cele mai multe din aplicaţiile grafice, în care scena este observată dintr-un singur punct de observare, matricea de observare $\mathbf{M}_V$ este unică pentru toate obiectele într-un cadru dat, dar se modifică la cadrul următor (imaginea următoare), dacă observatorul şi-a schimbat poziţia de observare, situaţie cea mai probabilă în grafica interactivă.

Matricea de normalizare $\mathbf{M}_N$ este definită de parametrii de proiecţie, care, aşa cum s-a prezentat în secţiunea precedentă, corespund construcţiei sistemului grafic (tipul şi numărul de proiectoare, unghiul de vizibilitate), deci această matrice este o caracteristică constructivă a sistemului grafic şi rămîne constantă pe toată perioada desfăşurării programului, pentru toate obiectele şi pentru toate imaginile generate (cadrele de imagine). În concluzie, sunt necesare următoarele matrice de transformare în cursul generării imaginilor succesive, la momentele (cadrele) i, i+1, i+2, …

Cadrul i:
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi}\mathbf{M}_{I1}$, pentru obiectul 1
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi}\mathbf{M}_{I2}$, pentru obiectul 2
.....
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi}\mathbf{M}_{Ik}$, pentru obiectul k
.....

Cadrul i+1:
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi+1}\mathbf{M}_{I1}$, pentru obiectul 1
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi+1}\mathbf{M}_{I2}$, pentru obiectul 2
.....
$\mathbf{M}=\mathbf{M}_N\mathbf{M}_{Vi+1}\mathbf{M}_{Ik}$, pentru obiectul k
.....

Se pune problema care este cea mai eficientă organizare a acestor operaţii de transformări succesive. Dacă s-ar utiliza o singură matrice curentă, secvenţa operaţiilor de compunere ar trebui să fie efectuată pentru fiecare obiect, în fiecare cadru de imagine. Este evident că utilizarea unei singure matrice curente de transformare este ineficientă şi nejustificată, dat fiind că o matrice este o resursă extrem de puţin costisitoare. Soluţia adoptată în OpenGL este de a se folosi mai multe stive de matrice de transformare, în care să fie reţinute şi reutilizate matricele intermediare de transformare.

Transformările de modelare şi de observare sunt efectuate într-o stivă de matrice, numită stiva matricelor de modelare-vizualizare (*modelview matrix stack*).

Transformarea de normalizare este prelucrată într-o stivă de matrice separată, numită stiva matricelor de proiecţie (*projection matrix stack*). Mai există încă o stivă de matrice pentru operaţiile de texturare, numită stiva matricelor de texturare (*texture matrix stack*). Separarea matricelor de modelare-vizualizare şi de proiecţie permite execuţia în paralel a operaţiilor din pipeline-ul grafic al acceleratoarelor hardware care au prevăzute resurse pentru astfel de prelucrări.

Fiecare stivă de matrice se comportă asemănător cu o stivă obişnuită, asupra căreia se operează prin funcţii de introducere (`push`) şi extragere (`pop`). Aceste funcţii diferă foarte puţin faţă de funcţiile `push` şi `pop` folosite în general în programare.

În orice moment, una din cele trei stive de matrice este declarată ca stivă curentă şi toate operaţiile cu matricele de transformare sunt adresate stivei curente. Setarea unei stive curente se face prin apelul funcţiei

```
void glMatrixMode(GLenum mode);
```

unde argumentul `mode` poate lua una din constantele simbolice `GL_MODELVIEW`, `GL_PROJECTION` sau `GL_TEXTURE` pentru setarea ca stivă curentă a stivei de modelare-vizualizare, a stivei de proiecţie sau a stivei de texturare

Matricea din vârful stivei curente este matricea curentă C, cu care se efectuează operaţiile cu matrice (`glLoadIdentity()`, `glTranslate#()`, `glLoadMatrix#()`, `glMultMatrix#()`, etc).

Funcţia:

```
void glPushMatrix();
```

adaugă o nouă matrice în stiva curentă, egală cu matricea din vârful acesteia, care devine noul vârf al stivei (fig. 6.4). Încărcarea unei noi valori în noua matrice curentă se face prin operaţii ulterioare de încărcare sau acumulare prin postmultiplicare. De fapt, aceasta este caracteristica cea mai importantă a stivelor matricelor de transformări, şi anume că operaţia `push` se execută prin două sau mai multe funcţii: `glPushMatrix()` creează o nouă poziţie în capul stivei, identică cu valoarea precedentă, şi în această poziţie se continuă operaţiile de compunere a matricelor prin postmultiplicare.

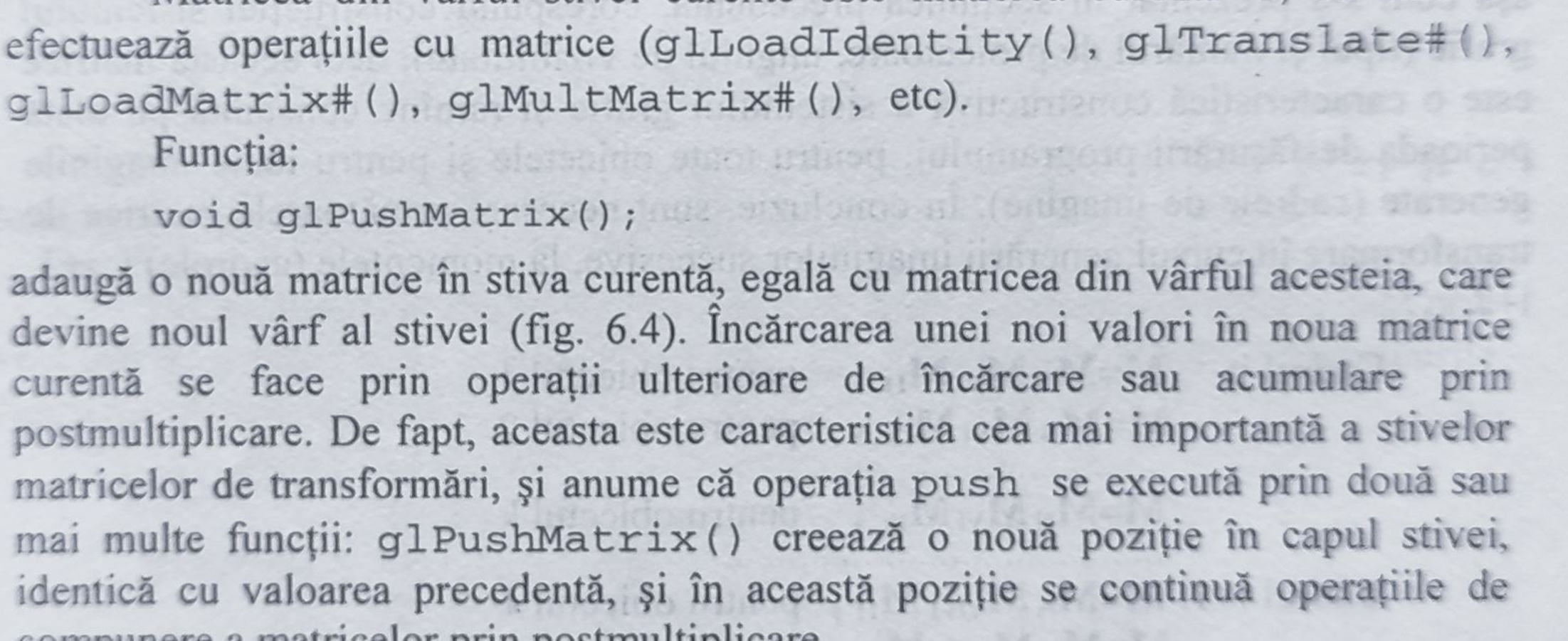


Fig. 6.4 Operaţii cu stiva matricelor de transformare.

Funcţia:

```
void glPopMatrix();
```

elimină matricea din vârful stivei curente, iar matricea precedentă acesteia devine noua matrice curentă. Această funcţie nu salvează matricea eliminată (deci ea nu este "extrasă", ca în operaţiile `pop` obişnuite), iar dacă această matrice ar mai fi necesară, ea trebuie copiată în altă zonă de memorie, înainte de apelul funcţiei `glPopMatrix()`.

Funcţia `glPushMatrix()` se apelează atunci când o matrice aflată în capul stivei trebuie salvată pentru a fi utilizată ulterior. Funcţia `glPopMatrix()` se apelează atunci când operaţiile de compunere a matricelor trebuie să continue cu o matrice aflată *sub* capul stivei. La iniţializarea bibliotecii OpenGL, fiecare stivă de matrice conţine o matrice care este vârful stivei respective.

Se poate prezenta acum un program OpenGL simplu, folosind utilitarul `GLUT`, care exemplifică folosirea stivelor matricelor de transformări geometrice.

■ Exemplul 6.3

Se reia ca exemplu de utilizare a stivelor matricelor de transformare programul din exemplul 4.2, în care se generează imaginea unei scene obţinute prin instanţierea a patru obiecte.

Matricea de normalizare şi proiecţie se calculează o singură dată şi se introduce în vârful stivei matricelor de proiecţie. Pentru aceasta, se setează ca stivă curentă stiva matricelor de proiecţie şi se iniţializează matricea din vârful acestei stive cu matricea identitate:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

Valoarea matricei de normalizare se setează folosind funcţia `glFrustum()` descrisă anterior. Funcţia `glFrustum()` creează matricea de normalizare corespunzătoare argumentelor de apel, o înmulţeşte cu matricea curentă (care are valoarea de matrice identitate) şi depune rezultatul în matricea curentă, care este matricea din vârful stivel matricelor de proiecţie. În acest exemplu se apelează:

```
glFrustum(-1, 1,-1, 1, 1, 40);
```

Matricele de transformare de modelare şi observare se compun în stiva matricelor de modelare-vizualizare. Sistemul de referinţă de observare are centrul în punctul (`xv, zv, yv`) şi este rotit cu un unghi `gama` în raport cu axa z. Paşii de calcul al matricelor sunt următorii:

(1) Mai întâi se setează ca stivă curentă stiva de modelare-vizualizare şi se iniţilalizează matricea din vârful acesteia cu matricea identitate:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

(2) Se creează în stiva curentă matricea de transformare de observare $\mathbf{M}_V = \mathbf{R}_Z(-\text{gama})\mathbf{T}(-xv, -yv, -zv)$:

```
glRotated(-gama, 0,0,1);
glTranslated(-xv,-zv,-zv);
```

(3) Primul obiect are matricea de instanțiere $\mathbf{M}_{II} = \mathbf{I}$, deci matricea curentă (care conține matricea de vizualizare) se aplică coordonatelor vârfurilor cubului din sistemul de referință de modelare. Funcția `Cube()` transmite aceste vârfuri către OpenGL.

(4) Dacă s-ar construi în acest moment matricea de transformare pentru obiectul al doilea, prin multiplicarea matricei curente cu matricele componente de instanțiere din $\mathbf{M}_2 = \mathbf{T}(0,-8,0)\ \mathbf{S}(2,2,2)$, atunci matricea de observare s-ar pierde şi pentru următorul obiect ar trebui calculată din nou. De aceea, se reține în stivă matricea de observare prin apelul funcției `glPushMatrix()`, care creează o nouă matrice în vârful stivei, cu aceeaşi valoare (egală cu $\mathbf{M}_V$). Matricea de instanțiere $\mathbf{M}_2$ se compune cu matricea de observare în această nouă matrice curentă, se apelează funcția `Cube()`, după care matricea din vârful stivei ($\mathbf{M}_V\mathbf{M}_2$) nu mai este necesară. Prin apelul funcției `glPopMatrix()`, această matrice este eliminată şi, în acelşi timp, se regăseşte în vârful stivei matricea de observare $\mathbf{M}_V$, care va putea fi folosită pentru obiectele următoare. Funcțiile apelate în acest pas sunt:

```
glPushMatrix();
   glTranslated(0,-8,0);
   glScaled(2,2,2);
   Cube();
glPopMatrix();
```

(5) Pentru celelalte obiecte se repetă operațiile de la pasul 4.

Programul următor, care conține cele explicate mai sus, este un exemplu tipic de program grafic folosind biblioteca OpenGL şi utilitarul GLUT. În acest program se poate urmări inițializarea bibliotecii GLUT, definirea funcțiilor `callback` şi modul de utilizare a stivelor de transformări geometrice. Unele funcții nou introduse în program sunt explicate în comentarii.

```
#include <gl/glut.h>
// Initializari
void Init(){
   glClearColor(1.0,1.0,1.0,0.0); // culoare stergere
   glEnable(GL_DEPTH_TEST);       // validare Z-buffer
}
// Redare cub
void Cube(){
   glColor3f(0.8,0.8,0.8);
   glBegin(GL_POLYGON);
      glVertex3d(-1,-1, 1);
      glVertex3d( 1,-1, 1);
      glVertex3d( 1, 1, 1);
      glVertex3d(-1, 1, 1);
glEnd();
```

```
        glColor3f(0,0,0);
        glBegin(GL_LINE_LOOP);
            glVertex3d(-1,-1,-1);
            glVertex3d( 1,-1,-1);
            glVertex3d( 1, 1,-1);
            glVertex3d(-1, 1,-1);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glVertex3d(-1,-1,-1);
            glVertex3d( 1,-1,-1);
            glVertex3d( 1,-1, 1);
            glVertex3d(-1,-1, 1);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glVertex3d(-1,1,-1);
            glVertex3d( 1,1,-1);
            glVertex3d( 1,1, 1);
            glVertex3d(-1,1, 1);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glVertex3d(-1,-1,-1);
            glVertex3d(-1, 1,-1);
            glVertex3d(-1, 1, 1);
            glVertex3d(-1,-1, 1);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glVertex3d(1,-1,-1);
            glVertex3d(1, 1,-1);
            glVertex3d(1, 1, 1);
            glVertex3d(1,-1, 1);
        glEnd();
}
// Generare axe de coordonate
void Axes(double x,double y, double z){
    glColor3f(0,0,0);
    glBegin(GL_LINES);
        glVertex3d(0,0,0);
        glVertex3d(x,0,0);
        glVertex3d(0,0,0);
        glVertex3d(0,y,0);
        glVertex3d(0,0,0);
        glVertex3d(0,0,z);
    glEnd();
    // Texte x, y, z
    glColor3ub(0, 0, 0);
    glRasterPos3f(1.1*x, 0.0, 0.0);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'x');
    glRasterPos3f(0.0, 1.1*y, 0.0);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'y');
    glRasterPos3f(0.0, 0.0, 1.1*z);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'z');
```

```
    // SAGETI
    ..........................................
}
// Functia apelata la schimbarea dim. ferestrei
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-(GLfloat)w/(GLfloat)h,
              (GLfloat)w/(GLfloat)h, -1, 1, 1, 40);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
static double xv = 2, yv = 2, zv = 20;
static double gama = 0;

// Functia callback de generare imagine
void Display(){
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
      glRotated(-gama, 0,0,1);
      glTranslated(-2,-2,-20);
      // transf. de obs.: C = Rz(-30)T(-2,-2,-20)

      Axes(16,8,16);
      Cube();                          // primul obiect

      glPushMatrix();                  // al doilea obiect
         glTranslated(0,-8,0);
         glScaled(2,2,2);
         // C = Rz(-30)T(-2,-2,-20)T(0,-8,0)S(2,2,2)
         Cube();
      glPopMatrix();                   // C= Rz(-30)T(-2,-2,-20)
      glPushMatrix();                  // al treilea obiect
         glTranslated(8,0,0);
         glRotated(45,0,0,1);
         glScaled(2,1,2);
         //C = Rz(-30)T(-2,-2,-20)T(8,0,0)Rz(45)S(2,2,2)
         Cube();
      glPopMatrix();                   // C= Rz(-30)T(-2,-2,-20)

      glPushMatrix();                  // al patrulea obiect
         glTranslated(-8,0,0);
         glRotated(-45,0,0,1);
         glScaled(2,1,2);
         //C=Rz(-30)T(-2,-2,-20)T(-8,0,0)Rz(-45)S(2,1,2)
         Cube();
      glPopMatrix();
   glPopMatrix();
   glutSwapBuffers();
```

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Transformari grafice");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

În funcția locală de inițializare `Init()` se setează culoarea de ştergere a bufferului de imagine folosind funcția `glClearColor()`. Această culoare de ştergere este memorată într-o variabilă de stare şi este folosită pentru ştergerea bufferului de imagine atunci când se apelează funcția `glClear()`. De asemenea, se validează testul de adâncime, pentru eliminarea suprafețelor ascunse.

Funcția cea mai importantă a programului este funcția `Display()` în care se execută toate operațiile de generare a unei imagini: ştergerea bufferului de imagine, (funcția `glClear()`), crearea matricelor de transformare în stiva matricelor de modelare-vizualizare, generarea imaginii a patru obiecte aplicând transformarea corespunzătoare modelului cub (funcția `Cube()`).

La sfârşitul operațiilor de generare a imaginii, se comută bufferul de imagine prin apelul funcției `glutSwapBuffers()`. Valoarea matricei curente în fiecare pas de execuție este scrisă în comentarii. Indentările instrucțiunilor nu sunt necesare, dar permit urmărirea mai uşoară a operațiilor cu stivele de matrice. Funcția callback `Display` este apelată de sistemul de operare ori de câte ori este necesară redesenarea ferestrei de afişare.

O formă foarte simplificată de desenare a unui obiect tridimensional format din mai multe fețe, fiecare față formată din mai multe vârfuri, este dată în funcția `Cube()`. Către OpenGL se transmit primitive geometrice compuse din vârfurile fiecărei fețe, date în coordonate în sistemul de referință local (de modelare). Dacă se urmăreşte diagrama de execuție a operațiilor în OpenGL din fig. 6.1, se vede că asupra vârfurilor unei primitive se aplică transformarea din matricea curentă din stiva de modelare-vizualizare. De aceea este posibil ca, pentru fiecare instanțiere, să fie transmise aceleaşi coordonate (ale modelului), care sunt transformate (instanțiere şi observare) în sistemul de referință de observare. În exemplul dat, prima față se desenează ca un poligon plin de culoare gri, iar celelalte, ca muchii de culoare neagră, pentru o mai bună percepere a imaginii desenate alb-negru.

Secvența de operații continuă cu: asamblarea primitivelor geometrice, folosind informațiile topologice transmise de funcția `glBegin()` şi vârfurile primitivei transformate în sistemul de referință de observare, normalizarea, folosind matricea din vârful stivei matricelor de proiecție, decuparea, împărțirea

omogenă, proiecţia, transformarea fereastră poartă şi conversia de rastru pentru înscrierea valorilor corespunzătoare a pixelilor primitivei geometrice.

Funcţia de redare a axelor de coordonate `Axes()` generează axele de coordonate şi numele lor, printr-un caracter afişat folosind funcţia `glutBitmapCharacter()`. Partea de desenare a săgeţilor axelor nu este prezentată, dar se poate completa cu uşurinţă.

Funcţia callback de redimensionare `Reshape()` primeşte la apelul ei de către sistem dimensiunile `w` şi `h` ale ferestrei de afişare. În această funcţie se setează dimensiunea porţii egală cu dimensiunea ferestrei de afişare şi cu colţul stânga-jos în punctul de coordonate (0,0) al ferestrei, prin apelul funcţiei `glViewport()`. Tot în această funcţie callback este definită transformarea de normalizare şi proiecţie perspectivă (`glFrustum()`) astfel încât scalarea dintre fereastra de vizualizare şi poarta de afişare (care a fost setată egală cu fereastra de afişare) să fie uniformă, adică raportul dintre dimensiunile pe orizontală şi verticală ale ferestrei de vizualizare să fie egal cu raportul dintre dimensiunile pe orizontală şi verticală ale porţii de afişare. Funcţia de definire a proiecţiei perspectivă şi a volumului de vizualizare, `glFrustum()` operează asupra stivei matricelor de proiecţie. Funcţia `Reshape()` este apelată atât la lansarea programului aplicaţiei, cât şi la orice modificare a dimensiunilor ferestrei.

În funcţia `main()` a programului aplicaţiei sunt apelate funcţiile GLUT de iniţializare a ferestrei de afişare, sunt înregistrate funcţiile callback necesare şi apoi se intră în execuţia programului la apelul funcţiei `glutMainLoop()`.

Imaginile din fig. 4.7 a,b,c,d,e,f s-au obţinut prin modificarea corespunzătoare a valorilor variabilelor xv, yv, zv, gamma şi recompilarea programului.

Stivele matricelor de transformare sunt deosebit de utile pentru redarea scenelor şi a obiectelor modelate ierarhic, în care scena sau fiecare obiect este compus din mai multe subobiecte, pe mai multe nivele de ierarhie. Fiecare obiect component este definit printr-o matrice de localizare relativ la obiectul căruia îi aparţine (obiect părinte), şi instanţierea acestuia se obţine prin compunerea matricei de localizare proprii cu matricea de localizare a obiectului părinte. Reprezentarea şi redarea scenelor modelate ierarhic este tratată în cap. 11, dar în exemplul următor se va prezenta un scurt program de aplicaţie care foloseşte compunerea mai multor transformări de modelare.

■ Exemplul 6.4

Programul descrie construirea unui sistem solar foarte simplificat, cu soarele, două planete şi un satelit al uneia dintre planete. Sferele care reprezintă aceste corpuri sunt redate folosind funcţiile `glutSolidSphere()` şi `glutWireSphere()` care primesc ca argumente dimensiunea, numărul de paralele şi numărul de meridiane ale sferei.

Se consideră că soarele se află în centrul sistemului de referinţă universal şi că planetele execută o mişcare de rotaţie în jurul soarelui în planul $z = 0$, la distanţe diferite. Satelitul execută o mişcare de rotaţie în jurul unei planete, de asemenea în

planul z = 0. Sistemul de referinţă de observare are axele paralele cu axele sistemului universal şi centrul pe axa z pozitivă, la o valoare convenabil aleasă pentru a avea obiectele în câmpul de vizibilitate.

Poziţia centrului unei planete depinde de raza de rotaţie r_P în jurul soarelui şi de unghiul de rotaţie (dat de variabila ρ_P care variază între 0 şi 360°) şi are coordonatele $C_P(r_P \cos\rho_P, r_P \sin\rho_P, 0)$. Se poate observa uşor că aceaste coordonate se obţin printr-o translaţie cu $\mathbf{T}(r_P,0,0)$, urmată de o rotaţie cu $\mathbf{R_Z}(\rho_P)$, deci matricea de instanţiere a unei planete este $\mathbf{M_P} = \mathbf{R_Z}(\rho_P)\mathbf{T}(r_P,0,0)$.

Transformarea de instanţiere a satelitului se calculează relativ la sistemul de referinţă local al planetei sale. La fel ca şi pentru o planetă, poziţia centrului satelitului depinde de raza de rotaţie (r_S) a acestuia şi unghiul de rotaţie (ρ_S), şi are coordonatele în sistemul de referinţă local al planetei $C_S(r_S \cos\rho_S, r_S \sin\rho_S, 0)$. Rezultă că matricea de instanţiere a satelitului relativ la sistemul de referinţă al planetei sale este $\mathbf{M_S} = \mathbf{R_Z}(\rho_S)\mathbf{T}(r_S,0,0)$. Programul de reprezentare a acestui sistem solar este următorul:

```
#include <gl/glut.h>
#define rp1 1.8
#define rp2 1.2
#define rs1 0.6
static int rop1 = 20, rop2 = 40, ros1 = 20;
static float zpos = 5;
void Init(){
   glClearColor(1.0,1.0,1.0,0.0);
   glEnable(GL_DEPTH_TEST);
}
void Axes(double x,double y, double z){
   // aceeaşi funcţie ca în exemplul precedent
}
void Keyboard(unsigned char key, int x, int y){
   switch (key){
   case 'Z':
      zpos = zpos+0.1;
      glutPostRedisplay();
      break;
   case 'z':
      zpos = zpos-0.1;
      glutPostRedisplay();
      break;
   case 'P':
      rop1 = (rop1+10)%360;
      rop2 = (rop2+10)%360;
      glutPostRedisplay();
      break;
case 'p':
      rop1 = (rop1-10)%360;
      rop2 = (rop2-10)%360;
      glutPostRedisplay();
```

```
            break;
        case 'S':
            ros1 = (ros1+10)%360;
            glutPostRedisplay();
            break;
        case 's':
            ros1 = (ros1-10)%360;
            glutPostRedisplay();
            break;
        }
}
void Display(void){
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
        glTranslatef(-1,-1,-zpos);          //transf.de obs.
        Axes(3,1.5,2.5);
        glColor3f(0.4,0.4,0.4);
        glutSolidSphere(0.6, 16, 16);       // soare

        glColor3f(0,0,0);
        glPushMatrix();
            glRotatef(rop1, 0,0,1);
            glTranslatef(rp1,0,0);
            glutWireSphere(0.25, 8, 8);     // planeta 1
            Axes(1,0.8,1.8);
            glRotatef(ros1, 0,0,1);
            glTranslatef(rs1,0,0);
            glutSolidSphere(0.1, 8, 8);     // satelit 1
            Axes(0.5,0.5,1);
        glPopMatrix();

        glRotatef(rop2, 0,0,1);
            glTranslatef(rp2,0,0);
            glutWireSphere(0.2, 8, 8);          // planeta 2
            Axes(0.5,1.1,2.5);
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60,(GLfloat)w/(GLfloat)h, 1, 40);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Sistem Solar");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMainLoop();
    return 0;
}
```

Imaginile obținute la execuția acestui program sunt prezentate în fig. 6.5. În funcția `Init()` se validează funcționarea algoritmului de Z-buffer, pentru ascunderea suprafețelor ascunse în funcție de adâncime (coordonata z în sistemul de referință normalizat) prin funcția `glEnable(GL_DEPTH_TEST)`. Valoarea cu care este şters bufferul de adâncime la începutul fiecărui cadru (la execuția funcției `glClear()`) are implicit valoarea maximă a coordonatei z a volumului canonic în sistemul de referință normalizat (1.0).

În funcția `Reshape()` este utilizată o altă funcție de definire a proiecției perspectivă, funcția `gluPerspective()` din biblioteca GLU, care este o versiune simplificată a funcției `glFrustum()`. Prototipul acestei funcții este:

```
void gluPerspective(GLdouble fovy, GLdouble aspect,
                    GLdouble zNear, GLdouble zFar);
```

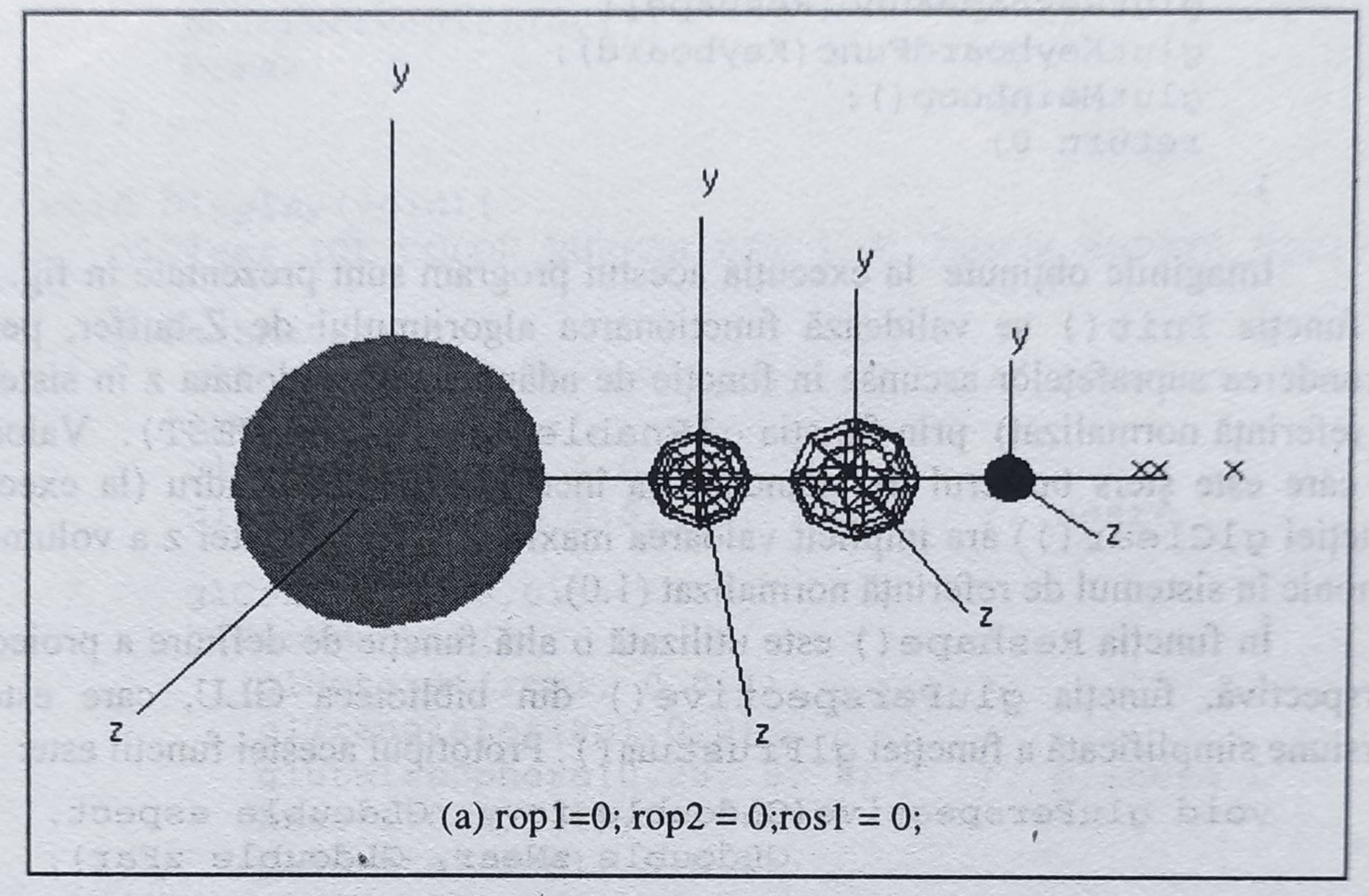
unde `fovy` este unghiul de vizibilitate pe verticală (în grade); `aspect` este raportul `w/h` a dimensiunilor pe orizontală şi verticală a ferestrei de afişare; `zNear` şi `zFar` sunt valorile absolute ale distanțelor planelor de vizibilitate apropiată, respectiv depărtată.

Funcția callback `Keyboard()` este apelată la acționarea unei taste. În acest exemplu simplu sunt tratate câteva taste. Tasta 'z' scade, iar tasta 'Z ' creşte valoarea coordonatei z_v a observatorului. Tastele 'p' şi 'P' scad, respectiv, cresc unghiul de rotație al planetelor. Tastele 's' şi 'S' scad, respectiv, cresc unghiul de rotație al satelitului. Se obține o mişcare interactivă (foarte simplificată, e adevărat, dar intuitivă) a planetelor şi satelitului.

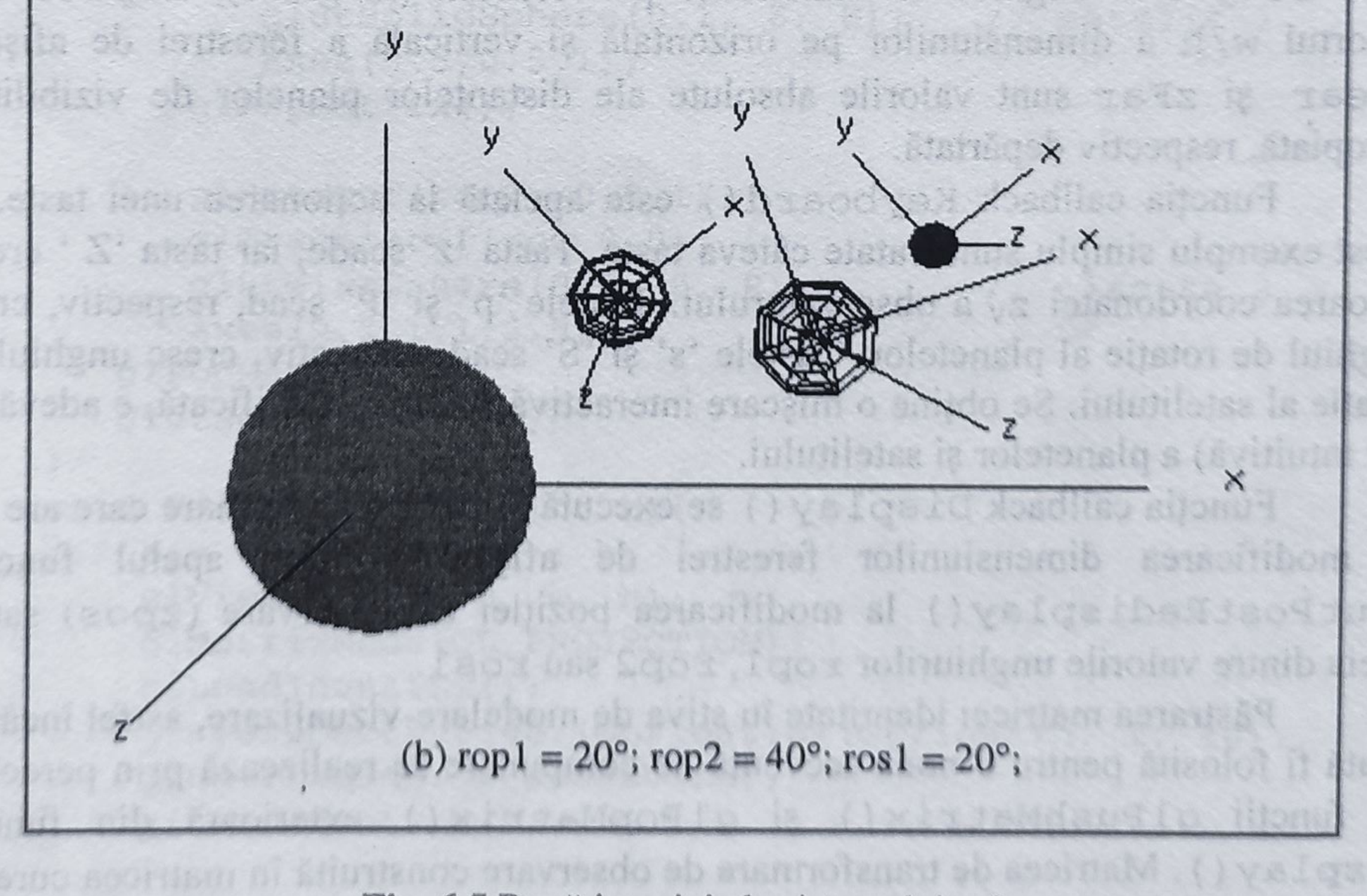
Funcția callback `Display()` se execută la fiecare redesenare care are loc la modificarea dimensiunilor ferestrei de afişare şi prin apelul funcției `glutPostRedisplay()` la modificarea poziției de observare (`zpos`) sau a uneia dintre valorile unghiurilor `rop1`, `rop2` sau `ros1`.

Păstrarea matricei identitate în stiva de modelare-vizualizare, astfel încât să poată fi folosită pentru o nouă secvență de compunere se realizează prin perechea de funcții `glPushMatrix()` şi `glPopMatrix()` exterioară din funcția `Display()`. Matricea de transformare de observare construită în matricea curentă este reținută în stivă prin cea de-a doua funcție `glPushMatrix()`. Matricea de instanțiere pentru prima planetă necesită introducerea mai întâi a rotației şi apoi a

translației și ea se aplică acesteia. Matricea de instanțiere a satelitului se construiește prin compunerea matricei sale de instanțiere cu matricea de instanțiere a planetei respective. După instanțierea satelitului, matricea de instanțiere a acestuia nu mai este necesară, funcția `glPopMatrix()`o elimină din stivă și vârful stivei conține acum matricea de transformare de observare, care poate fi folosită pentru compunerea matricei pentru cea de-a doua planetă.



(a) rop1=0; rop2 = 0;ros1 = 0;



(b) rop1 = 20°; rop2 = 40°; ros1 = 20°;

Fig. 6.5 Două imagini ale sistemului solar,
pentru două poziții diferite ale planetelor și satelitului.

În fig. 6.5 sunt prezentate două imagini captate din fereastra de afişare a programului pentru diferite valori ale poziţiilor planetelor şi satelitului. Imaginea nefiind color, s-a considerat că reprezentarea wireframe pentru planete este mai sugestivă. Pentru fiecare obiect s-au desenat şi axele de coordonate ale sistemul de referinţă local.

Este important de precizat faptul că, în marea majoritate a aplicaţiilor grafice, fiecare imagine (cadru, *frame*) se generează independent de imaginile precedente. Pentru fiecare cadru se calculează poziţia de observare şi poziţia obiectelor în scenă (dacă aceasta variază, aşa cum este situaţia în exemplul sistemului solar) şi imaginea se generează pornind de fiecare dată de la aceste informaţii. Trecerea de la o imagine la următoarea prin acumularea unor valori (de mişcare sau de poziţie) nu aduce nici un beneficiu semnificativ ca viteză de execuţie şi poate produce erori de poziţionare la execuţie un timp îndelungat. Chiar modificarea cu o valoare extrem de mică a poziţiei primitivelor geometrice redate pe display necesită ştergerea imaginii precedente şi construirea uneia noi prin reluarea întregii succesiuni a operaţiilor grafice, deoarece este foarte complicat (practic imposibil) de a se calcula ce părţi ale ecranului rămân neschimbate şi ce părţi se modifică şi cum se modifică.

6.4 DECUPAREA OBIECTELOR ÎN OPENGL

În OpenGL, decuparea primitivelor geometrice la volumul de vizualizare este efectuată în mod automat, fără nici o intervenţie din partea programatorului. Volumul de vizualizare se defineşte o dată cu definirea proiecţiei perspectivă (folosind una din funcţiile `glFrustum()` sau `gluPerspective()`) sau a proiecţiei paralele (folosind funcţia `glOrtho()`). Dacă se urmăreşte diagrama de execuţie din fig. 6.1, se poate observa că, după ce vârfurile au fost transformate prin înmulţire cu matricea curentă a stivei de modelare-vizualizare, se recompun primitivele geometrice, după care se aplică transformarea de normalizare şi decuparea. În cursul transformării de modelare-vizualizare, vârfurile au fost tratate individual (transformat fiecare în parte), dar pentru decupare este necesar să fie recompuse în primitive, deoarece decuparea se efectuează prin parcurgere în ordine a laturilor poligoanelor.

Fiind o bibliotecă de funcţii de nivel scăzut, care se ocupă de redarea unor primitive geometrice, OpenGL nu prevede nici o facilitate pentru eliminarea obiectelor care sunt în exteriorul volumului de vizualizare (*culling*). Este sarcina programatorului să definească volumele de delimitare, să le transforme din sistemul local în sistemul de referinţă de observare şi să le utilizeze pentru eliminarea obiectelor aflate complet în exteriorul volumului de vizualizare (obiecte invizibile).

Diferite sisteme de dezvoltare a aplicaţiilor grafice (cum sunt sistemele Sense8 şi Performer) oferă suport pentru definirea şi utilizarea volumelor de delimitare şi eliminare a obiectelor invizibile.

6.4.1 ELIMINAREA SUPRAFEȚELOR ASCUNSE

În OpenGL este implementat algoritmul Z-buffer de eliminare a suprafețelor ascunse. Numărul de biți/pixeli ai bufferului de adâncime se stabilește la definirea formatului pixelului. Implicit, se setează bufferul de adâncime cu 32 biți/pixel. Dacă se folosește utilitarul GLUT, atunci funcția de inițializare a modului de afișare trebuie să conțină opțiunea GLUT_DEPTH:

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
```

La inițializarea bibliotecii OpenGL testul de adâncime este invalidat. Pentru validarea lui se apelează funcția `glEnable(GL_DEPTH_TEST)`, iar la începutul fiecărui cadru de imagine trebuie să fie șters și bufferul de adîncime, o dată cu ștergerea bufferului de imagine, ceea ce se obține prin introducerea opțiunii `GL_DEPTH_BUFFER_BIT` în apelul funcției `glClear()`:

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Valoarea cu care se "șterge " bufferul de adâncime este implicit valoarea maximă 1, iar testul care se efectuează implicit este de înscriere a noului pixel dacă are adâncimea mai mică decât valoarea din buffer (`GL_LESS`). În OpenGL mai există posibilitatea de a defini altă valoare de ștergere a bufferului de adâncime (prin funcția `glClearDepth(GLclampd depth)`), ca și stabilirea testului de comparație, prin funcția

```
void glDepthFunc(GLenum func);
```

unde `func` poate fi una din mai multe constante simbolice admise (`GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS`) .

6.4.2 SELECȚIA SUPRAFEȚELOR ÎN FUNCȚIE DE ORIENTARE

OpenGL oferă suport direct pentru selecția primitivelor geometrice în funcție de orientarea lor. O primitivă geometrică este de categoria cu orientare directă (spre față - *frontface*) dacă sensul de parcurgere al acesteia (ordinea vârfurilor date prin funcțiile `glVertex#()` cuprinse în blocul `glBegin()`, `glEnd()`) este aceeași cu ordinea stabilită prin funcția `glFrontFace()` (în sensul acelor de ceas sau în sens invers acelor de ceas). Alftel, primitiva are orientare inversă (spre spate - *backface*). Prin funcția `glCullFace(GLenum type)` se selectează care tip de suprafețe trebuie eliminate: dacă argumentul `type` are valoare `GL_BACK` sunt eliminate primitivele cu orientare inversă; dacă are valoarea `GL_FRONT` sunt eliminate primitivele cu orientare directă. Testul de orientare și eliminarea primitivelor se execută numai dacă a fost validat acest test, prin apelul funcției glEnable(`GL_CULL_FACE`).

În segmentul de cod care urmează, cele două poligoane sunt cu orientare directă (front), deoarece vârfurile lor sunt date în sensul invers acelor de ceas, așa cum este setarea implicită a sensului de parcurgere directă în OpenGL. Primul

poligon este desenat, deoarece este activă opțiunea de eliminare a primitivelor cu orientare inversă, iar al doilea nu este desenat, deoarece este activă opțiunea de eliminare a primitivelor cu orientare directă.

```
void Display(){
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glEnable(GL_CULL_FACE);     // validează selectia
   glCullFace(GL_BACK); // se elimina primitive inverse
   glColor3f(0.5,0.5,0.5);
   glBegin(GL_POLYGON); //orientat direct, se deseneaza
      glVertex3d(3,3,-10);
      glVertex3d(4,3,-10);
      glVertex3d(4,4,-10);
      glVertex3d(3,4,-10);
   glEnd();
   glCullFace(GL_FRONT);// se elimina primitive directe
   glBegin(GL_POLYGON); //orientat direct, se elimină
      glVertex3d(1,1,-10);
      glVertex3d(2,1,-10);
      glVertex3d(2,2,-10);
      glVertex3d(1,2,-10);
   glEnd();
}
```

6.5 LISTE DE DISPLAY OPENGL

O listă de display este un grup de comenzi OpenGL care sunt memorate pentru a fi utilizate ulterior. Atunci când este apelată o listă de display, comenzile pe care le conține sunt executate în ordinea în care ele au fost memorate. Listele de display pot îmbunătăți performanțele de execuție atunci când se redesenează aceleaşi forme geometrice de mai multe ori.

Definirea unei liste de display începe cu apelul funcției `glNewList()` şi se termină la apelul funcției `glEndList()`. Prototipul funcția `glNewList` este:

```
void glNewList(GLuint list, GLenum mode);
```

Parametrul `list` este un indice care identifică lista de display şi este folosit la apelul acesteia. Parametrul `mode` poate lua una din constantele simbolice `GL_COMPILE` sau `GL_COMPILE_AND_EXECUTE`. Modul `GL_COMPILE` are ca efect compunerea listei de displaz. Modul `GL_COMPILE_END_EXECUTE` are ca efect compunerea listei şi execuția ei imediată. O listă de display cu indicele `list` este executată la apelul funcției:

```
void glCallList(GLuint list);
```

Ca exemplu de utilizare a unei liste de display se reiau comenzile de desenare a unui octogon din exemplul 6.2.

```
#include <math.h>
#include <gl\glut.h>
#define PI 3.141592
void DispList1(){
   glNewList(1,GL_COMPILE);
   int n = 8;
   double radius = 10;
   glColor3d(0,0,0);
   glBegin(GL_LINE_LOOP);
      for (int i=0;i<n;i++){
         double angle = 2*PI*i/n;
         glVertex3d(radius*cos(angle),
            radius*sin(angle),-40);
      }
   glEnd();
   glEndList();
}
 void Init(){
   glClearColor(1.0,1.0,1.0,1.0);
   DispList1();
}
 void  Display(){
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glCallList(1);
   glutSwapBuffers();
}
```

Funcțiile `Reshape()` şi `main()` ale programului sunt cele prezentate în programele precedente. Funcția `DisplayList()` apelată la inițializare crează lista de display cu indicele 1 care conține calculul coordonatelor octogonului şi OpenGL de desenare a acestuia. În funcția `Display()` este apelată funcția `glCallList(1)` care redesenează octogonul la fiecare apel.

Avantajul listelor de display este semnificativ atunci când, pentru generarea unei imagini, se folosesc funcții de calcul combinate cu comenzi OpenGL: calculele se execută o singură dată, la crearea listei, iar comenzile OpenGL rezultate se execută la fiecare apel al listei. O astfel de situație este prezentă şi în exemplul de mai sus şi apare frecvent în operațiile de iluminare, umbrire şi texturare a obiectelor.

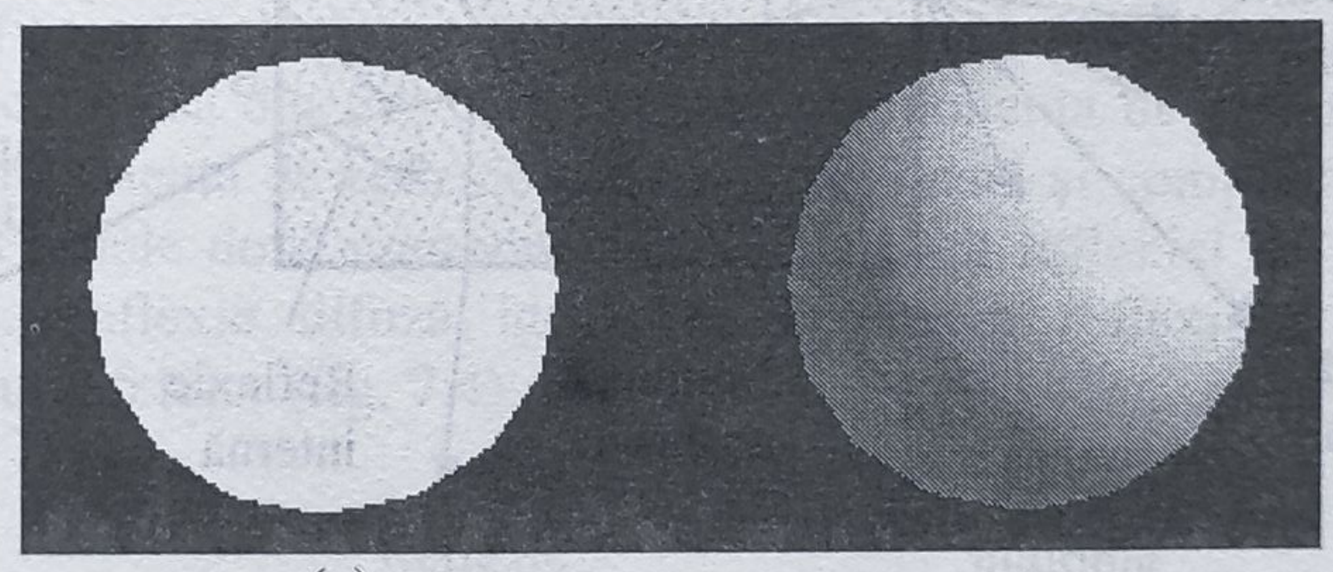
Dacă programul creează o singură listă de display, atunci se poate folosi valoarea 1 pentru indicele `list` din apelul funcției `glNewList()` aşa cum s-a procedat în exemplul precedent. Dacă se creează mai multe liste de display, indicii acestora sunt valori întregi succesive dintre care primul indice este obținut prin apelul funcției `glGenLists(GLsizei range)`.

Dacă valoarea returnată de funcția `glGenLists()` este numărul `k`, atunci indicii listelor care se pot crea au valorile `k`, `k+1`,....`k+range-1`. La primul apel al funcției `glGenLists()` se obține întotdeauna valoarea 1, deci în programele care creează o singură listă de display se poate folosi direct indicele 1 al listei.

7

MODELE DE REFLEXIE ŞI ILUMINARE

Redarea obiectelor tridimensionale prin suprafețe colorate uniform creează imagini nerealiste şi dificil de interpretat. De exemplu, dacă se redă imaginea unei sfere aproximate prin poligoane colorând fețele vizibile cu aceeaşi culoare albă, se obține un disc alb (fig. 7.1). Se poate vedea că o sferă neluminată arată la fel ca un disc bidimensional. Acest aspect nerealist apare datorită faptului că percepția celei de-a treia dimensiuni este mult influențată de modul de redare a iluminării obiectelor. Imaginea aceleiaşi sferei în care se ține seama de iluminare este mult mai sugestivă.



(a) (b)

Fig. 7.1 Imaginea unei sfere: (a) fără iluminare; (b) cu iluminare.

În lumea reală, atunci când lumina provenită de la diferite surse de lumină cade asupra obiectelor opace, o parte este absorbită de obiect, iar o parte este reflectată. Ochiul percepe lumina reflectată de obiect, pentru a interpreta forma, culoarea şi alte detalii ale obiectului. Pentru calculul iluminării în grafica pe calculator trebuie să fie definite sursele de lumină şi interacțiunea dintre lumină şi suprafețe.

Un *model de iluminare* defineşte natura luminii emise de o sursă de lumină, adică distribuția intensității luminii emise. Un *model de reflexie* descrie interacțiunea dintre lumină şi o suprafață, în funcție de proprietățile suprafeței şi natura sursei de lumină. Modelele de iluminare şi de reflexie în grafica pe

calculator permit redarea acceptabilă din punct de vedere al percepţiei umane a obiectele tridimensionale proiectate în spaţiul ecran bidimensional. Nivelul de acceptabilitate al redării depinde de natura aplicaţiei. Cu cât este cerut un nivel de realism mai ridicat, cu atât este necesar un model de reflexie mai complex şi cerinţe de prelucrare mai mari.

Implementarea unui model de reflexie în procedeul de calculare a intensităţii culorii fiecărui pixel este cunoscută sub numele de *tehnică de umbrire*.

7.1 CONSIDERAŢII TEORETICE ASUPRA REFLEXIEI LUMINII

Lumina incidentă la o suprafaţă a unui obiect este distribuită în patru categorii: lumina refectată, lumina adsorbită, lumina transmisă şi lumina împrăştiată şi emisă (fig. 7.2).

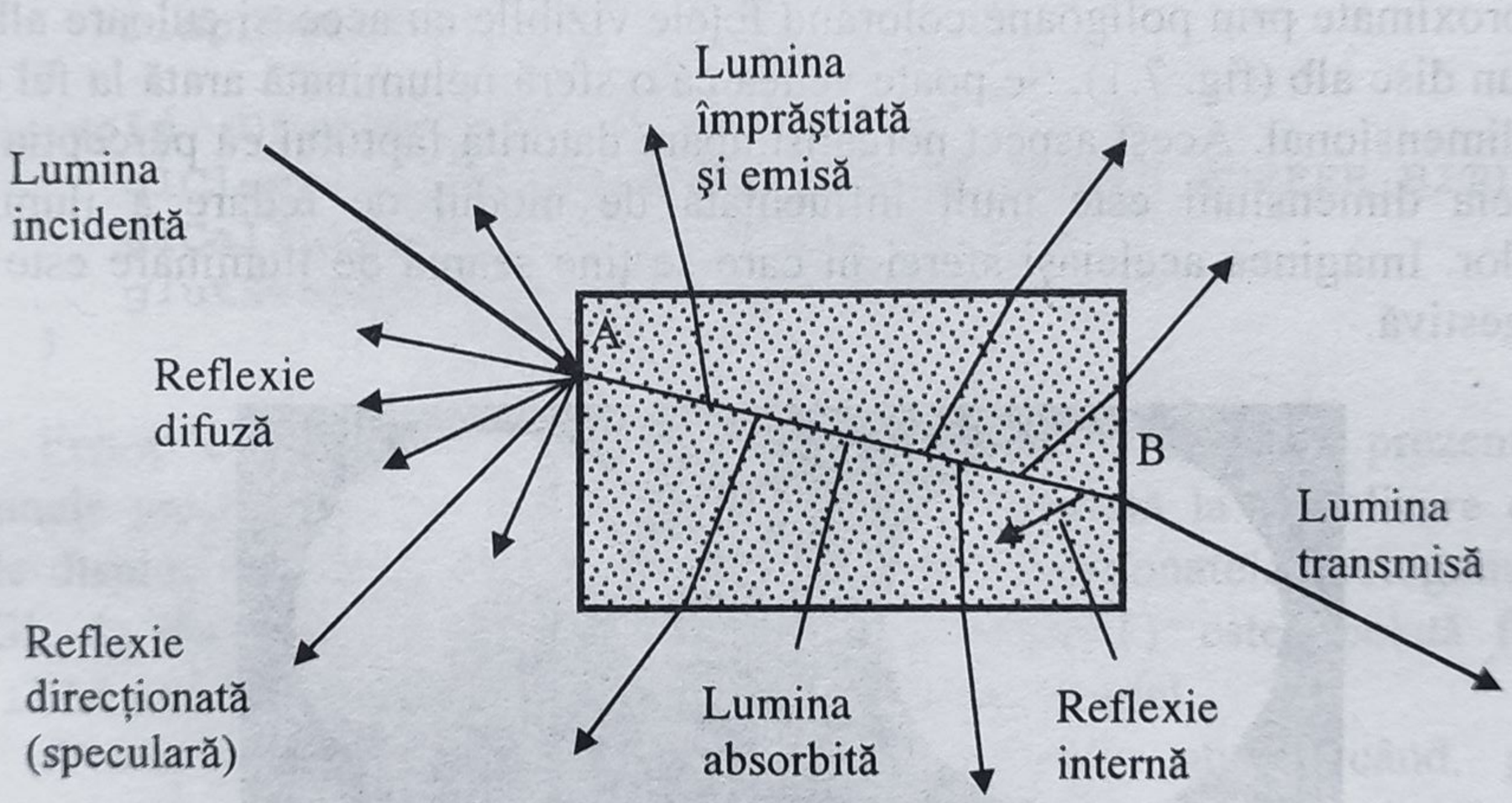


Fig. 7.2 Interacţiunea luminii cu un corp solid.

În sinteza de imagine se încearcă modelarea cât mai eficientă şi completă a acestui mod de interacţiune a luminii cu obiectele. Intensitatea şi lungimea de undă a luminii reflectate de o suprafaţă a unui obiect depinde de mai mulţi factori: lungimea de undă a luminii incidente, unghiul de incidenţă, natura suprafeţei şi proprietăţile ei electrice: permitivitatea, permeabilitatea şi conducţanţa. Modelul exact al interacţiunii este extrem de complex şi el poate fi aproximat prin intermediul *funcţiei de reflexie bidirecţională* (*bidirectional reflectivity function* – BDRF), care permite o apreciere cantitativă suficient de sugestivă pentru redarea iluminării obiectelor în grafica pe calculator.

Funcţia de reflexie bidirecţională este relaţia dintre intensitatea luminii reflectate în direcţia $(\phi_v \theta_v)$ şi energia luminii primite din direcţia $(\phi_i \theta_i)$ (fig. 7.3):

$$R_{bd}(\lambda, \phi_i, \theta_i, \phi_v, \theta_v) = \frac{I_V(\phi_i, \theta_i, \phi_v, \theta_v)}{E_i(\phi_i, \theta_i)} \tag{7.1}$$

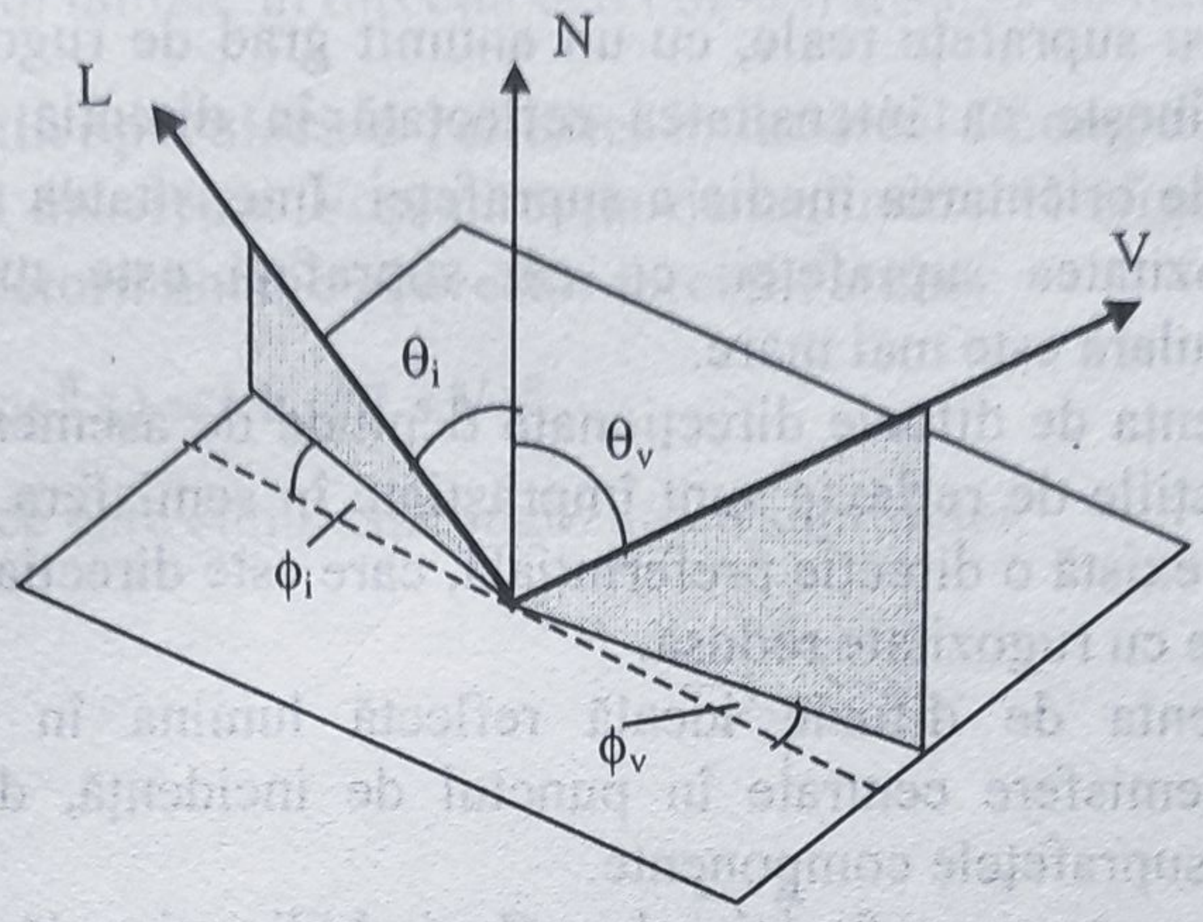


Fig. 7.3 Reflexia bidirecțională a luminii.

Funcția de reflexie bidirecțională depinde de lungimea de undă λ. Relația între energia luminii incidente corespunzătoare unui unghi solid ω_i şi intensitate I_i este:

$$E_i(\phi_i, \theta_i) = I_i(\phi_i, \theta_i)\cos\theta_i d\omega_i \tag{7.2}$$

În grafica pe calculator se consideră în mod simplificat că intensitatea reflectată este compusă din trei componente: componenta de reflexie direcționată (speculară), componenta de reflexie difuză direcționată şi componenta de reflexie difuză ideală. Primele două componente se datorează reflexiei de prim ordin, iar componenta de reflexie difuză ideală se datorează reflexiilor multiple şi a reflexiilor subsuprafețelor (fig. 7.4).

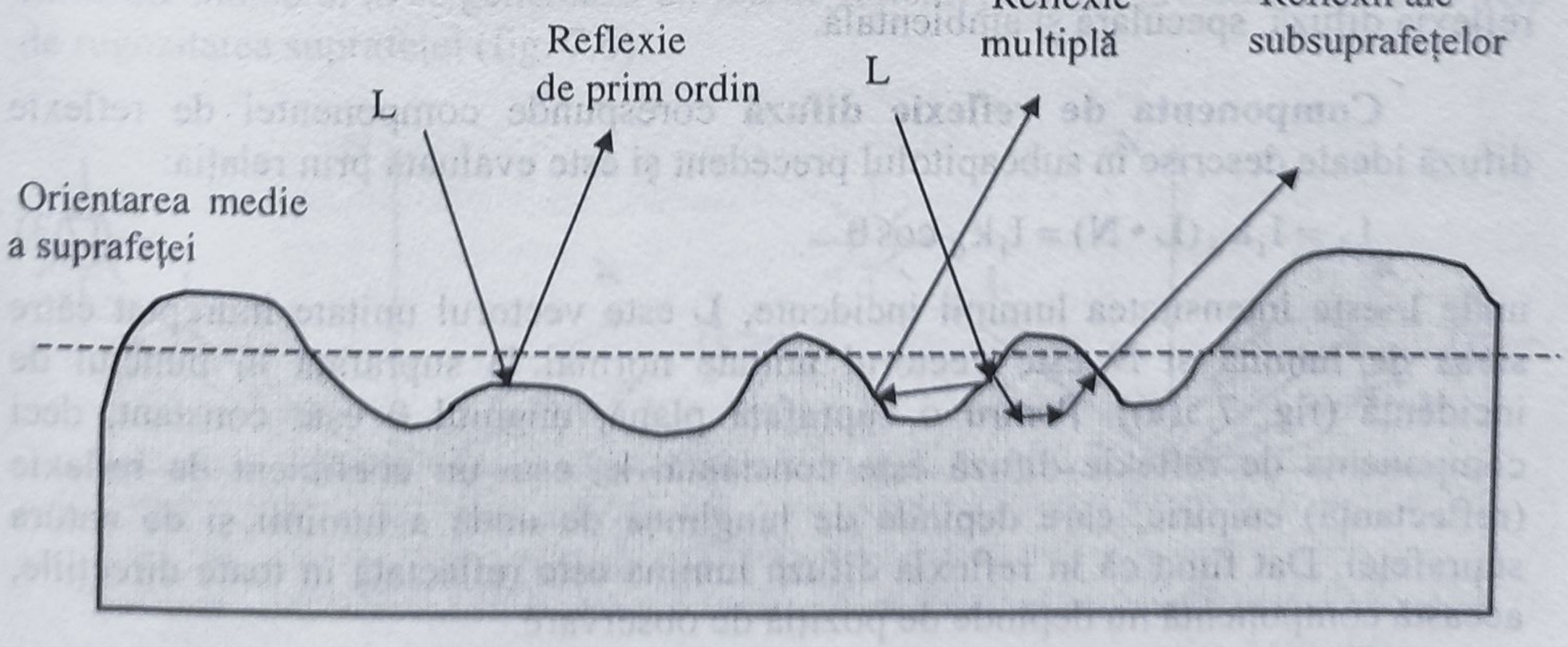


Fig. 7.4 Reflexii de prim ordin, reflexii multiple
şi reflexii ale subsuprafețelor

Valorile componentelor intensităţilor reflectate depind de rugozitatea suprafeţei şi de lungimea de undă a luminii. Dacă suprafaţa ar fi oglindă perfectă, atunci singura componentă de lumină reflectată ar fi componenta direcţionată (speculară). Pentru suprafeţe reale, cu un anumit grad de rugozitate, componenta speculară se defineşte ca intensitatea reflectată în direcţia: $\phi_v = \phi_i$, $\theta_v = \theta_i$, considerată faţă de orientarea medie a suprafeţei. Intensitatea acestei componente depinde de rugozitatea suprafeţei: cu cât suprafaţa este mai netedă, cu atât componenta speculară este mai mare.

Componenta de difuzie direcţionată depinde de asemenea de reflexiile de prim ordin. Direcţiile de reflexie sunt împrăştiate în semisfera centrată în punctul de incidenţă, dar există o direcţie preferinţială, care este direcţia reflexiei speculare pentru suprafeţele cu rugozitate redusă.

Componenta de difuzie ideală reflectă lumina în toate direcţiile în interiorul unei semisfere centrate în punctul de incidenţă, datorită împrăştierii provocate de subsuprafeţele componente.

Prin descompunerea funcţiei de reflexie bidirecţională în trei componente se poate dezvolta un model analitic bazat pe aspecte fizice, optice şi geometrice, care să permită simularea iluminării şi a reflexiei în grafica pe calculator. Modelul de reflexie Phong este cel mai cunoscut model în grafica pe calculator, care adoptă o reprezentare empirică şi fenomenologică, prin formule de calcul simple de imitare a comportării teoretice a reflexiei luminii, descrise anterior [Phong75].

7.2 MODELUL DE REFLEXIE PHONG

Modelul de reflexie Phong imită eficient modul real de reflexie, până la un grad care produce o percepţie destul de bună a obiectelor iluminate şi, de aceea, are o largă utilizare în grafica pe calculator. Modelul de reflexie Phong consideră lumina reflectată de o suprafaţă ca fiind o combinaţie liniară a trei componente: reflexia difuză, speculară şi ambientală.

Componenta de reflexie difuză corespunde componentei de reflexie difuză ideale descrise în subcapitolul precedent şi este evaluată prin relaţia:

$$I_d = I_i k_d (\mathbf{L} \bullet \mathbf{N}) = I_i k_d \cos\theta \qquad (7.3)$$

unde I_i este intensitatea luminii incidente, **L** este vectorul unitate îndreptat către sursa de lumină şi **N** este vectorul unitate normal la suprafaţă în punctul de incidenţă (fig. 7.5(a)). Pentru o suprafaţă plană, unghiul θ este constant, deci componenta de reflexie difuză este constantă. k_d este un coeficient de reflexie (reflectanţă) empiric, care depinde de lungimea de undă a luminii şi de natura suprafeţei. Dat fiind că în reflexia difuză lumina este reflectată în toate direcţiile, această componentă nu depinde de poziţia de observare.

Dacă există mai multe surse de lumină, atunci:

$$I_d = k_d \sum_n I_{i,n} (\mathbf{L}_n \bullet \mathbf{N}) \qquad (7.4)$$

unde $\mathbf{L}_n$ este vectorul unitate în direcţia celei de-a n-a surse de lumină.

Componenta specular㠹 a reflexiei în modelul Phong depinde de unghiul Ω între direcţia de observare **V** şi direcţia de oglindire **R** (fig. 7.5(b)). Dacă se consideră **R** şi **V** vectorii unitate ai acestor direcţii, atunci:

$$I_s = I_i k_s \cos^n \Omega = I_i k_s (\mathbf{R} \bullet \mathbf{V})^n \qquad (7.5)$$

unde n este un indice care simulează rugozitatea suprafeţei.

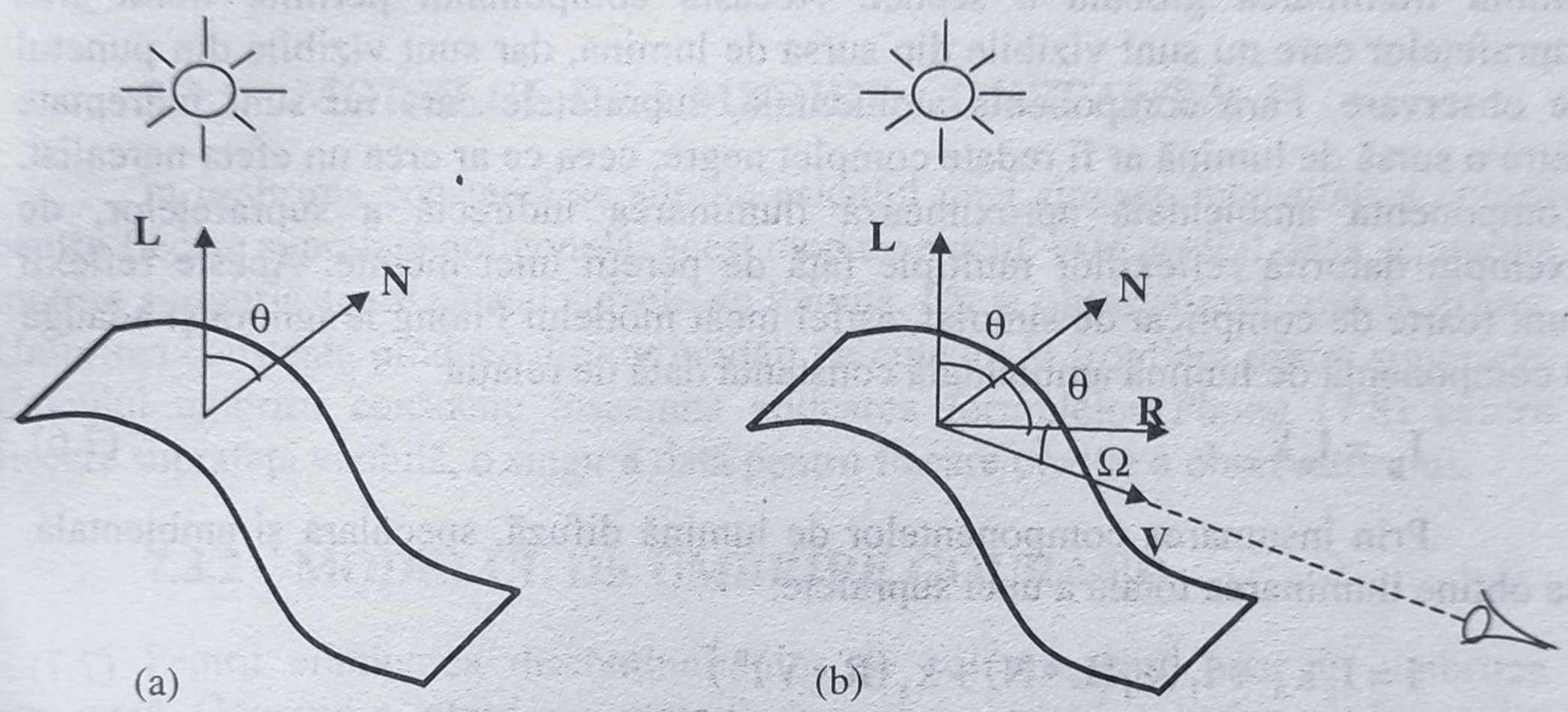


Fig. 7.5 (a) Reflexia difuză Phong. (b) Reflexia specular㠹 Phong.

Pentru o suprafaţă oglindă perfectă, n tinde către infinit şi lumina este reflectată numai în direcţia de oglindire R, pentru care $\cos^n \Omega = 1$. Pentru valori finite ale indicelui n, se generează un lob de reflexie, a cărui grosime este o funcţie de rugozitatea suprafeţei (fig. 7.6).

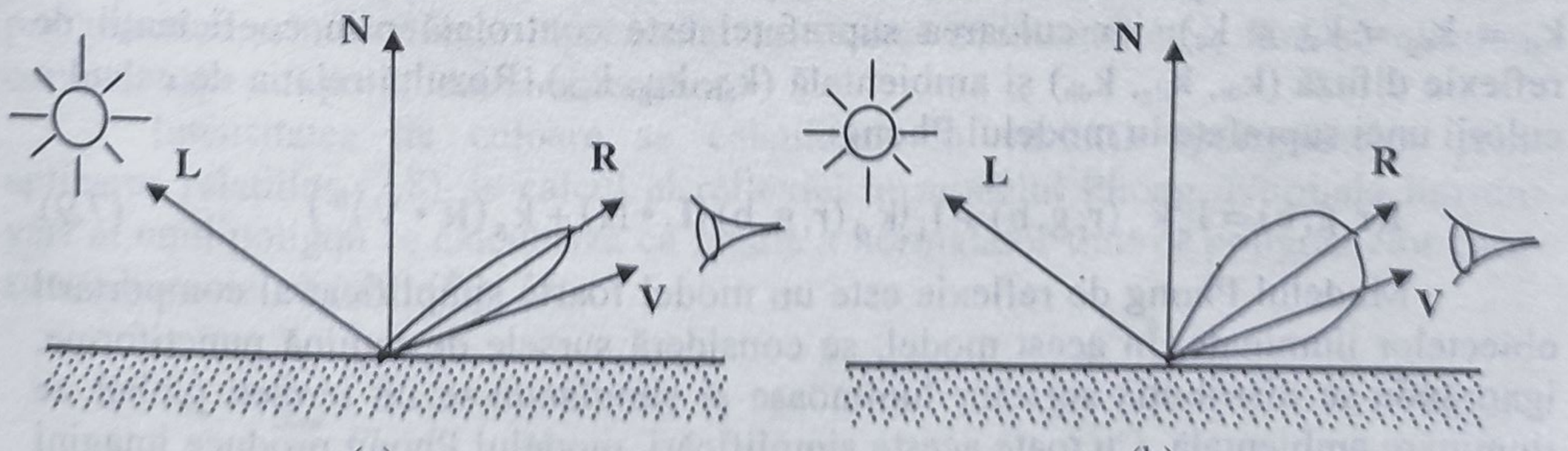


Fig. 7.6 (a) Indice n mare simulează reflexie specular㠹 în lob îngust.
(b) Indice n mic simulează reflexie specular㠹 în lob larg.

Efectul reflexiei speculare în modelul Phong este acela de a produce o iluminare mai accentuată (*highlight*), care este reflexia sursei de lumină pe o arie a suprafeței, depinzând de valoarea lui n (Planşa 1). Culoarea luminii reflectate specular poate fi diferită de culoarea luminii reflectate difuz. În modelele cele mai simple de reflexie specular, se presupune că această componentă are culoarea sursei de lumină. De exemplu, o suprafață de culoare verde iluminată de o sursă de lumină albă produce o componentă de reflexie de difuzie de culoare verde, dar pata luminoasă de reflexie specular are culoarea albă.

Componenta de lumină ambientală se adaugă în modelul Phong pentru a simula iluminarea globală a scenei. Această componentă permite iluminarea suprafețelor care nu sunt vizibile din sursa de lumină, dar sunt vizibile din punctul de observare. Fără componenta ambientală, suprafețele care nu sunt îndreptate către o sursă de lumină ar fi redate complet negre, ceea ce ar crea un efect nerealist. Componenta ambientală aproximează iluminarea indirectă a suprafețelor, de exemplu datorită reflexiilor multiple față de pereții unei incinte. Aceste reflexii sunt foarte de complicat de simulat, astfel încât modelul Phong le ignoră şi adaugă o componentă de lumină ambientală constantă dată de relația:

$$I_g = I_a k_a \tag{7.6}$$

Prin însumarea componentelor de lumină difuză, specular şi ambientală, se obține iluminarea totală a unei suprafețe:

$$I = I_a k_a + I_i \left(k_d (\mathbf{L} \bullet \mathbf{N}) + k_s (\mathbf{R} \bullet \mathbf{V})^n \right) \tag{7.7}$$

În modelul RGB de reprezentare a culorilor, relația (7.7) se descompune în trei relații, pentru fiecare componentă roşu, verde, albastru:

$$\begin{aligned} I_r &= I_a k_{ar} + I_i \left(k_{dr} (\mathbf{L} \bullet \mathbf{N}) + k_{sr} (\mathbf{R} \bullet \mathbf{V})^n \right) \\ I_g &= I_a k_{ag} + I_i \left(k_{dg} (\mathbf{L} \bullet \mathbf{N}) + k_{sg} (\mathbf{R} \bullet \mathbf{V})^n \right) \\ I_b &= I_a k_{ab} + I_i \left(k_{db} (\mathbf{L} \bullet \mathbf{N}) + k_{sb} (\mathbf{R} \bullet \mathbf{V})^n \right) \end{aligned} \tag{7.8}$$

În mod obişnuit se consideră componenta specular de culoare albă (deci $k_{sr} = k_{sg} = k_{sb} = k_s$), iar culoarea suprafeței este controlată prin coeficienții de reflexie difuză (k_{dr}, k_{dg}, k_{db}) şi ambientală (k_{ar}, k_{ag}, k_{ab}). Rezultă relația de calcul a culorii unei suprafețe în modelul Phong:

$$I(r,g,b) = I_a k_a (r,g,b) + I_i \left(k_d (r,g,b) (\mathbf{L} \bullet \mathbf{N}) + k_s (\mathbf{R} \bullet \mathbf{V})^n \right) \tag{7.9}$$

Modelul Phong de reflexie este un model foarte simplificat al comportării obiectelor iluminate. În acest model, se consideră sursele de lumină punctiforme, ignorându-se distribuția surselor luminoase şi simulându-se un termen global de iluminare ambientală. Cu toate aceste simplificări, modelul Phong produce imagini cu un grad de realism care este suficient pentru multe aplicații.

În Planşa 1 sunt reprezentate obiecte de culoare gri iluminate de surse de lumină de diferite intensități şi culori. Se poate observa efectul reflexiei de difuzie, specular şi ambientale asupra aspectului obiectelor.

7.3 MODELE DE UMBRIRE

Aplicarea directă a relației (7.8) pentru calculul culorii fiecărui pixel necesită un timp de execuție extrem de ridicat, care nu este acceptabil în grafica interactivă. De aceea, în sinteza de imagine se folosesc anumite metode simplificate de calcul al culorii fiecărui pixel, numite tehnici (modele) de umbrire. Tehnicile de umbrire depind de modul de reprezentare a obiectelor. Pentru obiectele modelate prin rețea de poligoane, se folosesc mai multe modele de umbrire: umbrirea constantă (poligonală, *flat*), umbrirea Gouraud şi umbrirea Phong.

7.3.1 MODELUL DE UMBRIRE CONSTANTĂ

În umbrirea constantă se admite calculul unei singure intensități a culorii pentru fiecare suprafață poligonală; acest mod de calcul este posibil dacă se admite ipoteza simplificatoare că atât sursele de lumină, cât şi observatorul sunt la infinit. Umbrirea constantă produce discontinuități de culoare la frontiera dintre suprafețe. Calculul umbririi constante înseamnă aplicarea formulelor Phong (7.8) pentru fiecare suprafață vizibilă, o singură dată pentru fiecare poziție a observatorului.

7.3.2 MODELUL DE UMBRIRE GOURAUD

Pentru eliminarea discontinuităților de colorare care apar în umbrirea constantă, Gouraud a introdus o metodă de umbrire care-i poartă numele şi care calculează intensitatea de culoare a pixelilor suprafețelor prin metode de interpolare, pornind de la intensitățile în vârfurile poligonului [Gour71]. Metoda incrementală de calcul al intensității de culoare a pixelilor este asemănătoare metodei de calcul al adâncimii pixelilor folosită în algoritmul Z-buffer. În general, se aplică combinat transformarea de rastru, eliminarea suprafețelor ascunse şi umbrirea Gouraud, într-un algoritm de baleiere pe linii generalizat.

Umbrirea Gouraud este o tehnică de interpolare biliniară a intensității culorii, foarte simplă şi economică, care atenuează discontinuitățile de la frontiera poligoanelor prin care este reprezentat un obiect tridimensional, fără să elimine complet aspectul poligonal al obiectelor.

Intensitatea de culoare se calculează în vârfurile poligoanelor prin aplicarea relațiilor (7.8) de calcul al reflexiei în modelul Phong. Normala într-un vârf al unui poligon se calculează ca medie a normalelor tuturor poligoanelor care sunt adiacente vârfului (fig. 7.7):

$$N_v = \sum_{i=1}^{m} \frac{N_i}{m} \tag{7.10}$$

Normalele în vârfurile poligoanelor sunt definite în sistemul de referință model şi ele sunt transformate în sistemul de referință de observare prin aplicarea aceloraşi transformări care se aplică vârfurilor: transformarea de instanțiere şi

transformarea de observare. În sistemul de referinţă de observare se calculează intensităţile în vârfurile poligoanelor şi aceste valori sunt folosite pentru interpolare biliniară în algoritmul de conversie de baleiere pe linii a poligoanelor.

Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, şi memorate ca parte a modelului obiectului în baza de date grafice. Atunci când se decupează feţele obiectelor, pot să apară vârfuri noi, care nu existau în modelul obiectului iniţial. Pentru aceste vârfuri se calculează normalele în noile vârfuri prin interpolare între normalele laturilor intersectate de planul de decupare.

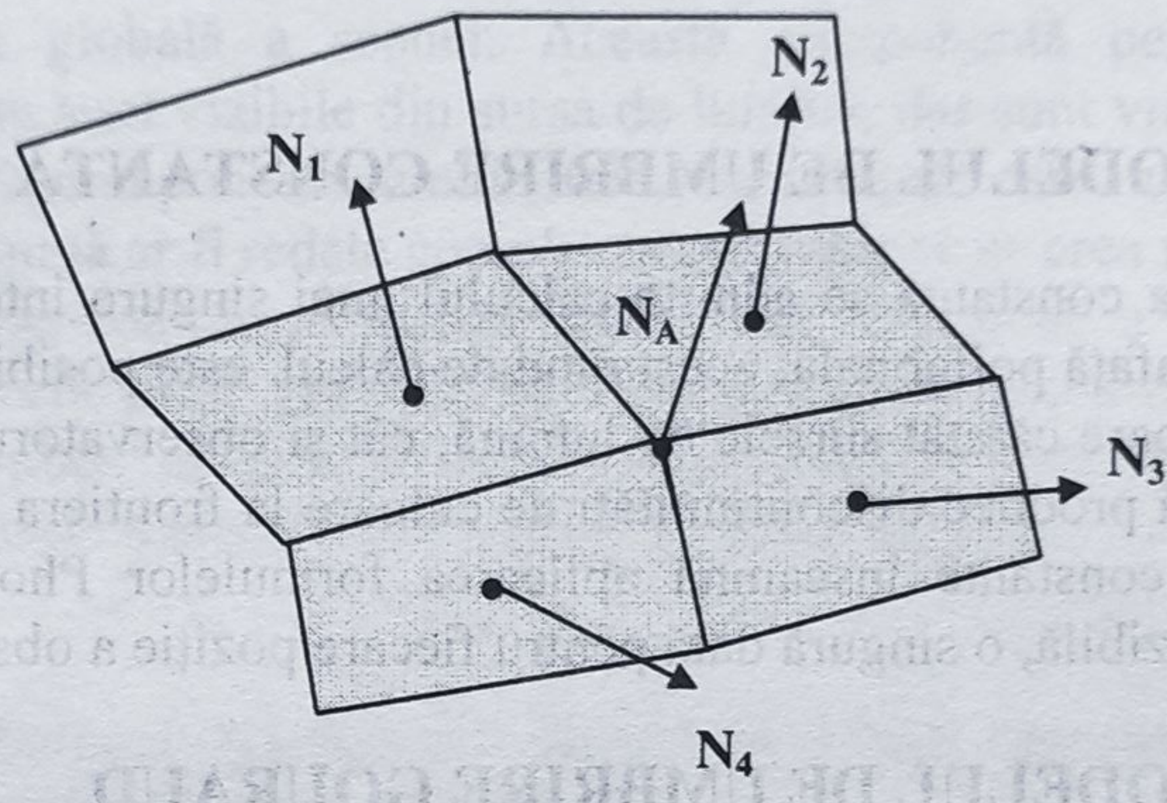


Fig. 7.7 Normala la vârful N_A este media normalelor N_1, N_2, N_3, N_4 a poligoanelor adiacente în vârful respectiv.

Interpolarea biliniară a intensităţilor de culoare a pixelilor unui poligon se execută în cadrul algoritmului de baleiere pe linii pornind de la intensităţile de culoare ale vârfurilor poligonului, calculate prin relaţiile modelului de reflexie Phong.

Se reia exemplul de baleiere pe linii din § 5.3 şi 5.4.3. Poligonul ABCDE are intensităţile de culoare calculate în vârfuri: I_A, I_B, I_C, I_D, şi, respectiv, I_E. Intensităţile la capetele liniilor de baleiere se calculează din intensităţile vârfurilor (fig. 7.8). Pentru laturile AB şi BC, ecuaţiile de interpolare sunt:

$$I = m_{i1} y + n_{i1}$$
$$I = m_{i2} y + n_{i2} \qquad (7.11)$$

unde:
$$m_{i1} = (I_A - I_B) / (y_A - y_B), \quad n_{i1} = y_B - m_{i1} I_B$$
$$m_{i2} = (I_C - I_B) / (y_C - y_B), \quad n_{i2} = y_C - m_{i2} I_C \qquad (7.12)$$

Din motive de eficienţă a calculelor, aceste ecuaţii se implementează incremental. Intensităţile se calculează în punctele de intersecţie cu liniile de baleiere y_1, y_2, ...y_i, y_{i+1},...Fiind calculate intensităţile $I_{i,1}$ şi $I_{i,2}$ corespunzătoare liniei de baleiere $y = y_i$, intensităţile $I_{i+1,1}$ şi $I_{i+1,2}$, corespunzătoare liniei de baleiere următoare, $y_{i+1} = y_i + 1$ se obţin prin incrementare:

$$I_{i+1,1} = I_{i,1} + m_{i1}$$
$$I_{i+1,2} = I_{i,2} + m_{i2} \qquad (7.13)$$

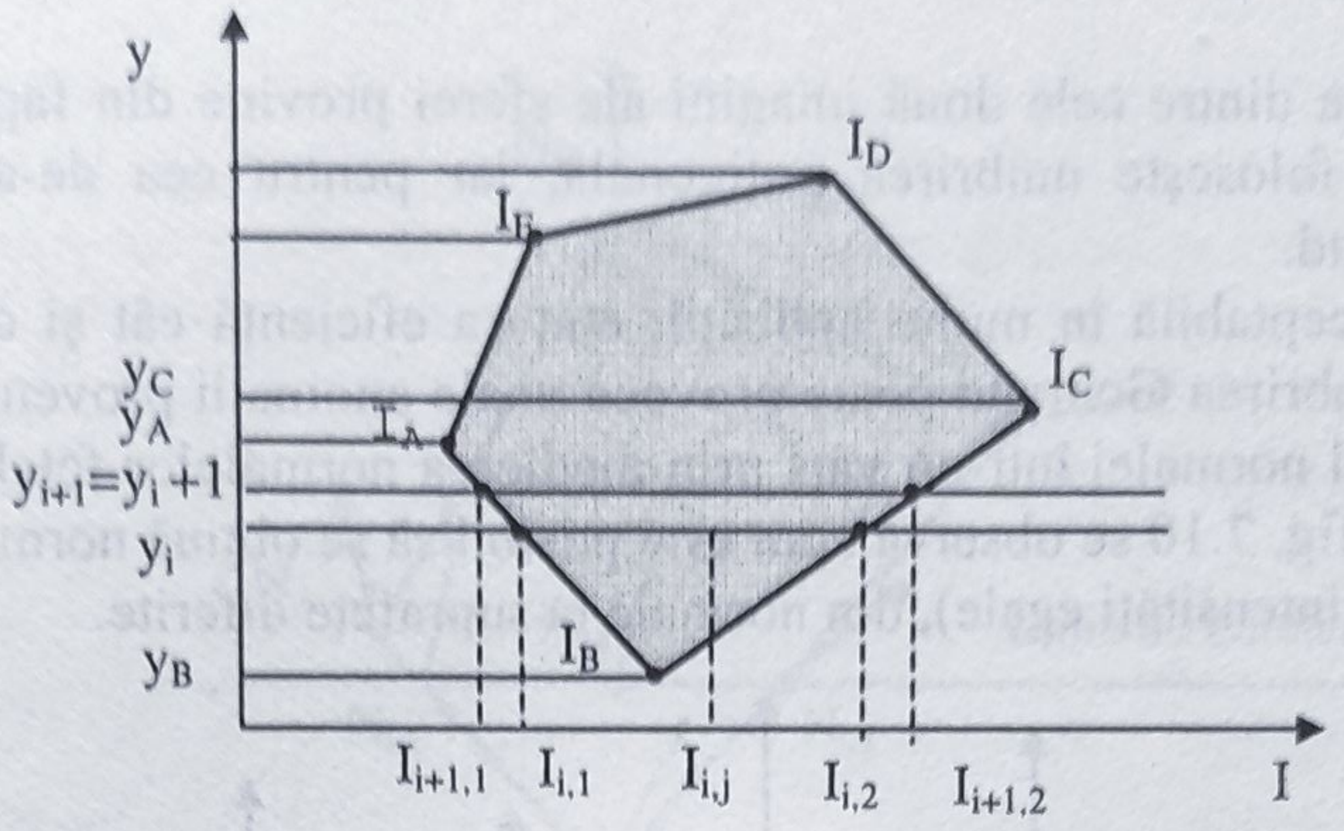


Fig. 7.8 Interpolarea intensității pe laturile poligonului și pe o linie de baleiere în umbrirea Gouraud.

Intensitatea $I_{i,1}$ se atribuie pixelului a cărui adresă se obține din coordonatele punctului $(x_{i,1}, y_i)$, intensitatea $I_{i,2}$ se atribuie pixelului a cărui adresă se obține din coordonatele punctului $(x_{i,2}, y_i)$, etc. Intensitățile de-a lungul liniei de baleiere se calculează din intensitățile la capetele acesteia. Pentru linia de baleiere $y = y_i$, ecuația de interpolare a intensității este:

$$I = m_{ix}x + n_{ix}$$

unde: $m_{ix} = (I_{i,2} - I_{i,1}) / (x_{i,2} - x_{i,1}), \quad n_{ix} = x_{i,1} - m_{ix} I_{i,1}$ (7.14)

Implementarea eficientă a acestei operații se face de asemenea incremental. Fiind calculată intensitatea $I_{i,j}$ corespunzătoare punctului de coordonate $(x_{i,j}, y_i)$, intensitatea în punctul următor eşantionat pe linia de baleiere $(x_{i,j+1}, y_i$, unde $x_{i,j+1} = x_{i,j}+1)$ este:

$$I_{i,j+1} = I_{i,j} + m_{ix} \quad (7.15)$$

Pentru fiecare linie de baleiere se calculează intensitatea în puncte succesive prin incrementarea valorii precedente cu coeficientul dat de relația (7.14).

În fig. 7.9 este exemplificată diferența dintre umbrirea poligonală şi umbrirea Gouraud. Două sfere reprezentate prin rețea de suprafețe plane (poligoane) sunt iluminate de la aceeaşi sursă punctiformă de lumină şi au aceleaşi caracteristici de reflexie ale materialului.

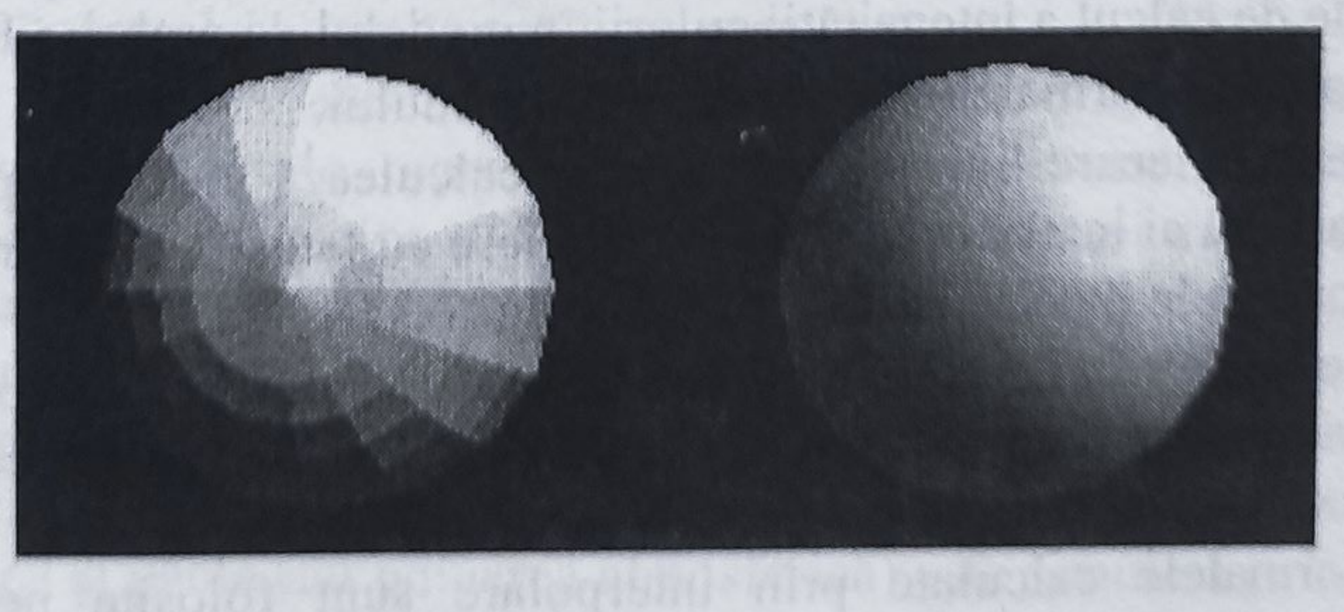


Fig. 7.9 Diferența dintre umbrirea poligonală şi umbrirea Gouraud.

Diferența dintre cele două imagini ale sferei provine din faptul că pentru prima sferă se folosește umbrirea poligonală, iar pentru cea de-a doua sferă, umbrirea Gouraud.

Deşi acceptabilă în multe aplicații, atât ca eficiență cât şi ca realism de reprezentare, umbrirea Gouraud poate provoca unele anomalii provenind în primul rând din calculul normalei într-un vârf prin medierea normalelor fețelor adiacente. De exemplu, în fig. 7.10 se observă cum este posibil să se obțină normale în vârfuri identice (şi deci intensități egale), din normale la suprafețe diferite.

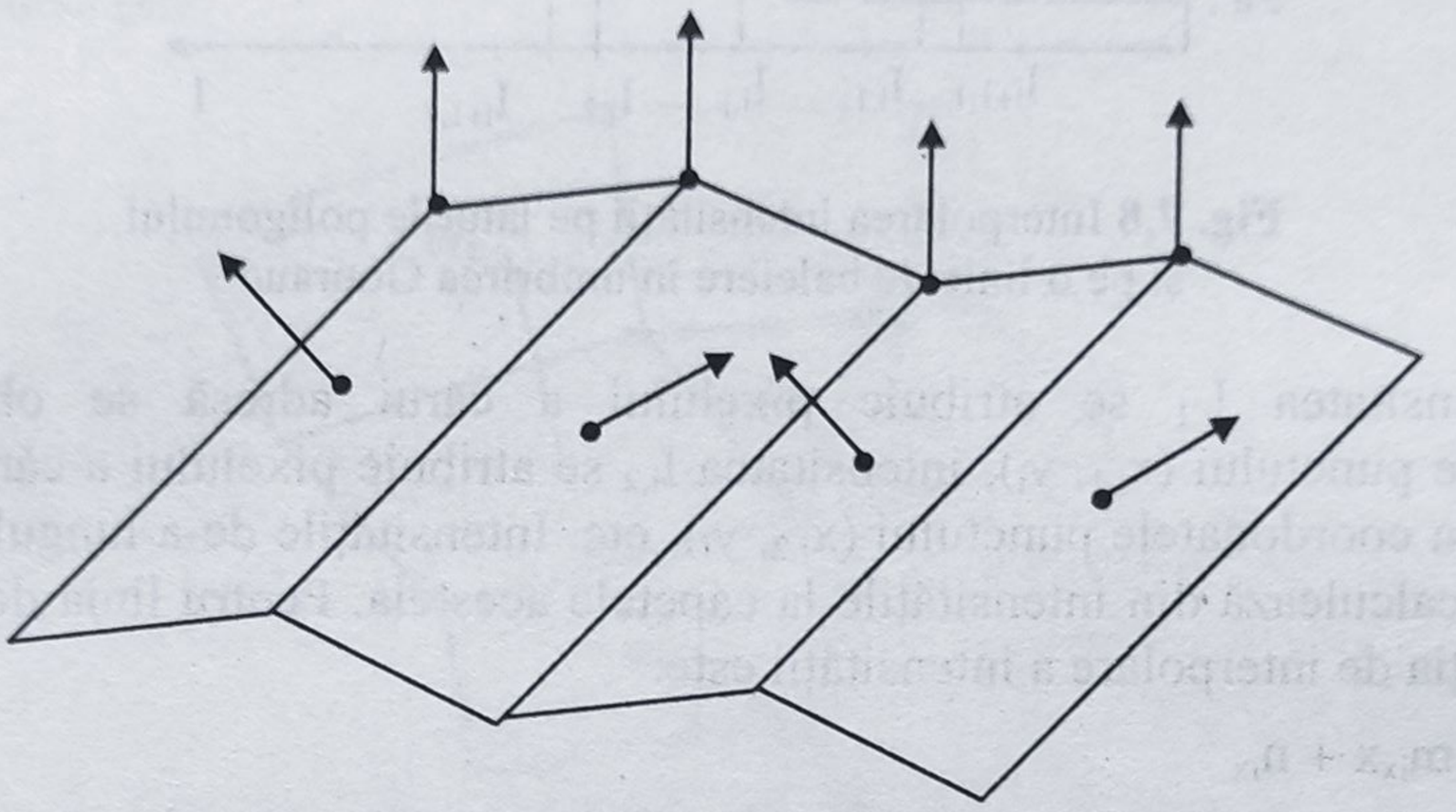


Fig. 7.10 O suprafață pliată regulat produce normale în vârfuri egale, deci colorare constantă a suprafețelor în umbrirea Gouraud

O altă deficiență a modelului Gouraud este aceea că nu se poate obține o pată de lumină (*highlight*) datorată reflexiei speculare în interiorul unui poligon, dacă vârfurile poligonului nu sunt cuprinse în această pată de lumină.

O parte din deficiențele tehnicii de umbrire Gouraud sunt eliminate în modelul de umbrire Phong.

7.3.3 MODELUL DE UMBRIRE PHONG

Modelul de umbrire Phong este, ca şi modelul Gouraud, un model de calcul al intensității culorii prin interpolare biliniară, dar în acest model se interpolează normalele şi se calculează exact intensitatea culorii (fig. 7.11).

Etapele de calcul a intensității culorii în modelul de umbrire Phong sunt:

- Calculul normalelor în vârfurile poligonului.
- Pentru fiecare linie de baleiere, se calculează prin interpolare vectorii normali ai intersecției liniei de baleiere cu laturile poligonului, folosind normalele în vârfuri ($N_{i,1}$ şi $N_{i,2}$ în fig. 7.11).
- Normalele la capetele unei linii de baleiere sunt folosite pentru calculul prin interpolare a normalei fiecărui punct (căruia în corespunde un pixel) de pe linia de baleiere ($N_{i,j}$).
- Normalele calculate prin interpolare sunt folosite pentru calculul intensității culorii în fiecare punct.

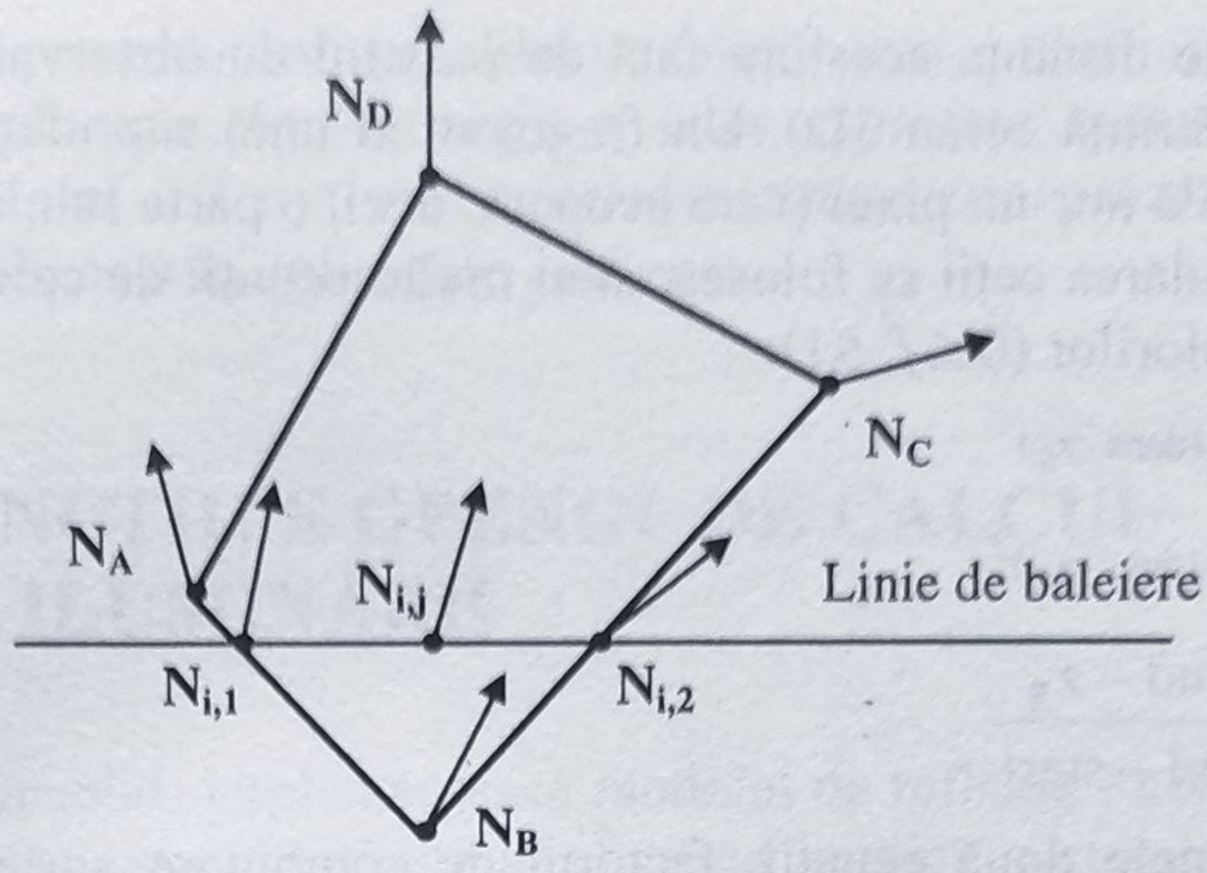


Fig. 7.11 Interpolarea normalelor în modelul de umbrire Phong.

Umbrirea Phong este mai costisitoare din punct de vedere al operațiilor efectuate deoarece, pentru fiecare pixel, se aplică relația de calcul (7.8), care conține produse scalare de vectori şi înmulțiri. Dar, din punct de vedere al calității imaginii generate, umbrirea Phong este mai bună, dat fiindcă se calculează normale foarte apropiate de normala reală în fiecare punct al suprafeței. În tehnica de umbrire Phong se poate simula mult mai corect reflexia speculară (*highlight*).

În subcapitolul următor sunt prezentate modalitățile de programare a iluminării în aplicații grafice folosind biblioteca OpenGL.

7.4 GENERAREA FENOMENELOR NATURALE

Obiectele tridimensionale (clădiri, vehicule, etc) sunt uşor de modelat şi de redat în grafica pe calculator şi sunt intens folosite în scenele virtuale sau în sistemele de proiectare CAD. În aplicațiile de realitate virtuală în care scena este un spațiu geografic, realismul imaginii necesită însă şi reprezentarea unor obiecte şi fenomene naturale: teren, copaci, apă, foc, fum, nori, ceață, pâclă, poluarea atmosferei, etc., fără de care imaginea se prezintă supărător de artificială. Sinteza obiectelor şi a fenomenelor naturale este mult mai dificilă decît cea a obiectelor tridimensionale "normale", cu o formă bine precizată.

Pentru simularea efectelor atmosferice precum *ceață*, *pâclă*, *poluare atmosferică*, modalitatea de reprezentare cea mai frecvent folosită este prin modificarea culorilor obiectelor din scenă pe baza distanței față de punctul de observare. Această operație este, de cele mai multe ori, implementată hardware în procesul de transformare de rastru a primitivelor geometrice. Pentru alte obiecte sau fenomene naturale există o varietate de reprezentări ad-hoc, care încearcă să obțină maximum de realism cu consum de resurse cât mai redus.

Ceața (ca şi pâcla sau poluarea atmosferică) este simulată prin combinarea culorii ceții cu culoarea fragmentelor suprafețelor, folosind un factor de combinare

care depinde de distanţa acestora faţă de punctul de observare (coordonata z_S în sistemul de referinţă ecran 3D). Un fragment al unei suprafeţe este porţiunea din suprafaţă vizibilă într-un pixel (care acoperă, deci, o parte sau întreg pixelul).

În simularea ceţii se folosesc mai multe ecuaţii de calcul al factorului f de combinare a culorilor ($0 \le f \le 1$):

$$f = e^{-(dens \cdot z_S)} \tag{7.16}$$

$$f = e^{--(dens \cdot z_S)^2} \tag{7.17}$$

$$f = \frac{end - z_S}{end - start} \tag{7.18}$$

În primele două ecuaţii, factorul de combinare variază exponenţial, iar parametrul *dens* (un număr pozitiv) poate fi variat pentru obţinerea unor efecte diferite de ceaţă. În cea de-a treia ecuaţie, factorul de combinare variază liniar cu distanţa de la suprafaţă la punctul de observare, între limitele *start* (de unde "începe" ceaţa) pînă la limita *end* (unde "se termină" ceaţa).

Fiecare componentă C_r a culorii unui fragment (C poate fi R, G, B, A) se calculează prin combinarea culorii C_s a fragmentului cu culoarea C_f a ceţii, prin ecuaţia:

$$C_r = fC_s + (1 - f)C_f \tag{7.19}$$

În combinarea liniară a culorilor pentru simularea ceţii, la limita maximă *end*, nu se mai distinge nici un obiect, toate având culoarea unică a ceţii. În general, limita *end* trebuie să fie corelată cu distanţa planului de vizibilitate depărtat (*far*) din definirea sistemului de vizualizare (trunchiul de piramidă de vizualizare). Dat fiind că nu se generează imaginea nici unui obiect aflat la distanţă mai mare decât distanţa maximă de vizibilitate, limita *end* trebuie să fie mai mică sau egală cu distanţa de vizibilitate maximă (*far*).

Introducerea fenomenului de ceaţă este necesară în special în scenele virtuale care modelează arii geografice în care se desfăşoară simularea antrenamentelor de zbor sau de conducere a altor vehicule. Impresia creată de obiecte clar desenate chiar atunci când sunt la distanţe mari este nerealistă, iar simularea ceţii îmbunătăţeşte imaginea generată, făcând-o mai apropiată de imaginea reală, în care obiectele aflate la distanţă mare se văd estompate. În acest mod, simularea ceţii contribuie la percepţia distanţei obiectelor în scenă (*depth cueing*).

În funcţie de parametrii de simulare a ceţii, se pot obţine diferite alte efecte. De exemplu, ceaţa de densitate redusă şi care apare numai la distanţe foarte mari, capătă aspect de pâclă; ceaţa de densitate redusă, distribuită uniform în scenă, poate fi considerată poluare atmosferică.

Un alt avantaj al simulării ceţii îl reprezintă diminuarea efectului de aliasing şi de aliasing al texturilor. Prin combinarea culorii suprafeţelor cu culoarea ceţii toate suprafeţele sunt estompate şi zomotul de aliasing, chiar dacă este prezent, este mai puţin supărător. În cazul texturării, dat fiind că efectul de aliasing este mai pregnant atunci când suprafaţa este observată în perspectivă (deoarece

creşte dimensiunea pre-imaginii pixelilor), estomparea culorilor cu creşterea distanţei atenuează efectele de "moaraj" a unei suprafeţe texturate privite în perspectivă. În exemplul 7.8 din subcapitolul următor este prezentat modul de obţinere a efectului de ceaţă în aplicaţiile grafice.

7.5 FUNCŢIILE OPENGL DE CALCUL AL ILUMINĂRII

Biblioteca OpenGL implementează modelul de reflexie Phong şi modelele de umbrire poligonală şi umbrire Gouraud. În mod implicit, sistemul de iluminare este inactivat şi pentru desenarea primitivelor geometrice se foloseşte culoarea curentă, specificată prin funcţia `glColor#()`. Pentru validarea iluminării obiectelor, se activează sistemul de iluminare OpenGL prin apelul funcţiei `glEnable (GL_LIGHTING)`.

Pentru calculul iluminării obiectelor trebuie să fie definite:

- sursele de lumină;
- materialul suprafeţelor;
- modelul de umbrire a suprafeţelor.

7.5.1 DEFINIREA SURSELOR DE LUMINĂ

În OpenGL se pot defini mai multe surse de lumină punctiforme. Numărul de lumini admise variază în funcţie de implementare, dar cel puţin opt lumini sunt disponibile în orice bibliotecă OpenGL. Fiecare sursă de lumină poate fi validată prin apelul funcţiei `glEnable(GL_LIGHTi)`, unde i este indexul sursei de lumină.

O sursă de lumină se caracterizează prin intensitate şi poziţie în scenă. Intensitatea unei surse de lumină se specifică pentru fiecare componentă de iluminare (ambientală, de difuzie şi speculară) printr-un vector în spaţiul culorilor în modelul RGBA. Poziţia unei surse se specifică printr-un vector în coordonate omogene corespunzătoare sistemului de referinţă universal.

Funcţia de definire a unui parametru al unei surse de lumină este funcţia `glLight#()` care are mai multe variante în funcţie de tipul argumentelor. De exemplu:

```
void glLightfv(GLenum light, GLenum pname,
               const GLfloat *params);
void glLightiv(GLenum light, GLenum pname,
               const GLint *params);
```

Argumentul `light` reprezintă indexul sursei de lumină şi poate lua un nume simbolic de forma `GL_LIGHT0, GL_LIGHT1, ....GL_LIGHTi`, unde $0 \leq i <$ `GL_MAX_LIGHTS`. Numărul maxim de lumini depinde de implementarea bibliotecii.

Argumentul pname specifică un parametru al sursei de lumină. Sunt acceptate mai multe valori, dintre care unele se referă la intensitate de culoare iar altele la poziţia sursei.

Valorile GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR ale argumentului pname permit definirea componentelor intensităţii culorii sursei de lumină. În această situaţie, argumentul params reprezintă un pointer la un vector de patru valori (de tip întreg sau virgulă flotantă), care specifică intensităţile RGBA ale componentei de iluminare ambientală, difuză şi, respectiv, specular. Valorile implicite ale intensităţilor sunt (0.0, 0.0, 0.0, 1.0) pentru componenta ambientală şi pentru componentele difuză şi specular a oricărei lumini cu excepţia luminii cu index 0, care au intensitatea difuză şi specular implicită (1.0, 1.0, 1.0, 1.0).

În acest model de definire a sursei de lumină, componenta ambientală (GL_AMBIENT) se referă la intensitatea RGBA pe care o sursă de lumină o adaugă iluminării globale a scenei. Componenta de iluminare difuză (GL_DIFFUSE) este cel mai apropiat mod de reprezentare a ceea ce se consideră culoarea sursei de lumină. Componenta de iluminare specular (GL_SPECULAR) afectează culoarea zonei strălucitoare (*hightlight*) a obiectelor luminate de sursa respectivă. Valoarea transparenţei surselor de lumină (*alpha*) este ignorată dacă nu se validează calculul transparenţei (*color blending*).

Valoare GL_POSITION a argumentului pname permite definirea poziţiei sursei de lumină. În această situaţie, argumentul params este un pointer la un vector de patru numere (întregi sau virgulă flotantă) care reprezintă poziţia în coordonate omogene în sistemul de referinţă universal a sursei de lumină. Acestă poziţie este transformată prin aplicarea valorii matricei de modelare-vizualizare din momentul apelului funcţiei glLight#(), astfel încât sursa de lumină va avea coordonatele transformate în sistemul de referinţă de observare, unde se calculează intensitatea în vârfurile primitivelor geometrice.

Dacă componenta w a poziţiei este 0, atunci lumina este tratată ca o lumină direcţională plasată la infinit, în direcţia definită de componentele x, y, z şi se utilizează această direcţie pentru calculul componentelor de reflexie difuză şi specular.

Dacă $w \neq 0$, atunci sursa este o sursă poziţională şi se foloseşte localizarea acesteia pentru calculul direcţiei de iluminare a suprafeţelor. Poziţia implicită a unei surse este (0, 0, 1, 0), deci sursa este plasată la infinit pe axa z şi ea luminează obiectele în direcţia – z. Secvenţa de instrucţiuni pentru definirea intensităţii şi poziţiei sursei de lumină de indice 0 arată astfel:

```
GLfloat light_ambient[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

În iluminarea reală, intensitatea luminii primite de un obiect scade odată cu creşterea distanţei acestuia faţă de sursa de lumină. Dat fiind că sursele de lumină direcţionale (care au w =0) se consideră plasate la infinit, nu are sens calculul atenuării cu distanţa, astfel încât atenuarea este invalidată pentru astfel de surse. Pentru sursele poziţionale, se foloseşte un factor de atenuare cu valoarea:

$$f = \frac{1}{k_c + k_l d + k_q d^2} \qquad (7.20)$$

unde: d = distanţa dintre sursa de lumină şi vârful în care se calculează iluminarea
k_c = GL_CONSTANT_ATTENUATION
k_l = GL_LINEAR_ATTENUATION
k_q = GL_QUADRATIC_ATTENUATION

Implicit, aceşti parametri au valorile: $k_c = 1$, $k_l = 0$, $k_q = 0$, dar ei pot fi setaţi prin apelul uneia din funcţiile `glLightf()` sau `glLighti()` astfel:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Dacă se folosesc mai multe surse de lumină, pentru fiecare dintre ele se definesc componentele intensităţii, poziţia şi caracteristicile de atenuare.

7.5.2 DEFINIREA PROPRIETĂŢILOR MATERIALELOR

Pentru calculul intensităţii culorii unei suprafeţe, trebuie definiţi coeficienţii de reflexie pentru componentele de reflexie ambientală, difuză şi speculară. Aceste proprietăţi sunt considerate proprietăţi de material al suprafeţei şi se specifică prin apelul uneia sau mai multora din cele patru variante ale funcţiei `glMaterial#()`:

```
void glMateriali(GLenum face,GLenum pname,GLint param);
void glMaterialf(GLenum face,GLenum pname,
                 GLfloat param);
void glMaterialiv(GLenum face,
                  GLenum pname,GLint *param);
void glMaterialfv(GLenum face,GLenum pname,
                  GLfloat *param);
```

În aceste funcţii, argumentul `face` este numele feţei şi depinde de orientarea acesteia; poate lua ca valori constantele simbolice GL_FRONT, GL_BACK, GL_FRONT_AND_BACK. Argumentul `pname` specifică proprietatea materialului, care se defineşte prin apelul funcţiei `glMaterial#()`. Acest argument poate lua una valorile date în tabelul 7.1.

Funcţiile `glMateriali()` şi `glMaterialf()` se folosesc numai pentru setarea strălucirii (GL_SHININESS) la valoarea dată prin argumentul `param`.

Funcţiile `glMaterialiv()` şi `glMaterialfv()` se folosesc pentru specificarea celorlalte proprietăţi. Dacă argumentul `pname` este `GL_COLOR_INDEX` atunci `param` este un pointer la un vector de trei valori de tip întreg, care conţine indicii culorilor ambientală, de difuzie şi specularã ale materialului. În celelalte situaţii, `param` este un pointer la un vector de patru valori de tip întreg sau cu virgulă flotantă, care sunt componentele roşu, verde, albastru şi alpha (transparenţă) ale componentei de reflexie specificate prin parametrul `pname` (ambientală, difuză, specularã, emisie). Se poate observa faptul că modelul OpenGL de reflexie permite adăugarea unei componente emisive a luminii reflectate de o suprafaţă.

Tabelul 7.1

Parametrii de definire a materialelor

Nume parametru	Valoare implicită	Semnificaţie
`GL_AMBIENT`	(0.2, 0.2, 0.2, 1.0)	Reflectanţa (coeficient de reflexie) ambientală
`GL_DIFFUSE`	(0.8, 0.8, 0.8, 1.0)	Reflectanţa de difuzie
`GL_AMBIENT_AND_DIFFUSE`		Reflectanţa ambientală şi de difuzie
`GL_SPECULAR`	(0.0, 0.0, 0.0, 1.0)	Reflectanţa specularã
`GL_EMISSION`	(0.0, 0.0, 0.0, 1.0)	Intensitatea luminii emise
`GL_SHININESS`	0.0	Exponentul de reflexie specularã
`GL_COLOR_INDEX`	(0, 1, 1)	Indicii culorilor ambientală, difuză şi specularã ale materialului

Reflexia difuză joacă cel mai important rol în culoarea pe care o prezintă o suprafaţă. Ea reprezintă culoarea pe care o are suprafaţa luminată direct şi depinde de componenta de difuzie a luminii incidente, provenită de la una sau mai multe surse de lumină, de coeficientul de reflexie de difuzie (reflectanţa de difuzie) a materialului şi de unghiul dintre direcţia luminii şi normala la suprafaţă. Poziţia de observare nu influenţează componenta de difuzie a luminii reflectate.

Reflexia ambientală afectează culoarea de ansamblu pe care o prezintă o suprafaţă şi ea devine sesizabilă atunci când suprafaţa nu este luminată direct. Ca şi reflexia difuză, această componentă nu depinde de poziţia de observare. Cele două componente se specifică de cele mai multe ori cu aceeaşi culoare (aşa cum sunt suprafeţele reale) folosind parametrul `GL_AMBIENT_AND_DIFFUSE` în apelul funcţiei `glMaterial#()`.

Reflexia specularã produce iluminarea mai puternică (*highlight*) a unei zone a obiectului, în funcţie de poziţia de observare. OpenGL permite specificarea culorii produse de reflexia specularã (prin parametrul `GL_SPECULAR`) şi a dimensiunii şi strălucirii zonei prin parametrul `GL_SHININESS`.

Componenta de emisie a unei suprafețe se specifică prin parametrul GL_EMISSION. Acest efect este folosit pentru simularea lămpilor sau a altor surse de lumină din scenă.

Proprietăților materialelor. Funcția glMaterial#() definește proprietățile materialului curent, care se aplică tuturor vârfurilor introduse după aceasta prin funcțiile glVertex#() sau prin diferite funcții de modelare din biblioteca GLUT (de exemplu, glutSolidSphere()). Proprietățile materialului curent se mențin până la următorul apel al funcției glMaterial#(). Calculele de iluminare se pot executa diferit pentru fețele orientate direct (GL_FRONT) și cele orientate invers (GL_BACK). Un exemplu de definire a unui material este dat prin următoarele instrucțiuni:

```
GLfloat mat_ambient[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat mat_shininess = 50.0;
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
```

Modelul de umbrire se definește prin apelul funcției glShadeModel(GLenum mode), unde argumentul mode poate lua una din valorile GL_FLAT, pentru modelul de umbrire poligonală, sau GL_SMOOTH, pentru modelul de umbrire Gouraud. Valoarea implicită este GL_SMOOTH.

Dacă nu este validat sistemul de iluminare (prin apelul funcției glEnable(GL_LIGHTING)) atunci culoarea care se atribuie vârfurilor primitivelor geometrice este culoarea curentă, setată prin apelul unei funcții glColor#().

Dacă s-a definit modelul de umbrire poligonală (GL_FLAT), primitivele geometrice se generează de culoare constantă (culoarea curentă întâlnită la primul vârf introdus prin funcția glVertex#()). Dacă s-a definit modelul de umbrire Gouraud (GL_SMOOTH), atunci culorile definite în vârfuri se folosesc pentru calculul intensității culorii pixelilor primitivei prin interpolarea biliniară.

■ Exemplul 7.1

În fig 7.12 (a) este reprezentat un dreptunghi cu umbrire Gouraud (GL_SMOOTH), iar în fig 7.12 (b) este reprezentat același dreptunghi cu umbrire poligonală (GL_FLAT), toate celelate funcții fiind aceleași.

Într-un mod asemănător, în Planșa 2 este reprezentat cubul culorilor RGB. Imaginile sunt obținute prin atribuirea culorilor roșu (001), verde (010), albastru (010), cian (011), magenta (101), galben (110), negru (000) și alb (111) vârfurilor cubului cu aceleași coordonate și prin validarea umbrii (glShadeModel (GL_SMOOTH)).

```
glShadeModel(GL_SMOOTH);
glBegin(GL_POLYGON);
    glColor3f(0.5,0.5,0.5);
    glVertex3d(-1,-1,-8);
    glColor3f(0.1,0.1,0.1);
    glVertex3d(1,-1,-8);
    glColor3f(0.5,0.5,0.5);
    glVertex3d(1,1,-8);
    glColor3f(0.9,0.9,0.9);
    glVertex3d(-1,1,-8);
glEnd();
```

(a)

```
glShadeModel(GL_FLAT);
glBegin(GL_POLYGON);
    glColor3f(0.5,0.5,0.5);
    glVertex3d(-1,-1,-8);
    glColor3f(0.1,0.1,0.1);
    glVertex3d(1,-1,-8);
    glColor3f(0.5,0.5,0.5);
    glVertex3d(1,1,-8);
    glColor3f(0.9,0.9,0.9);
    glVertex3d(-1,1,-8);
glEnd();
```

(b)

Fig. 7.12 (a) Interpolarea biliniară a culorilor din vârfurile primitivei. (b) Colorare constantă a primitivei.

Într-un mod asemănător, în Planşa 2 este reprezentat cubul culorilor RGB. Imaginile sunt obținute prin atribuirea culorilor roşu (001), verde (010), albastru (010), cian (011), magenta (101), galben (110), negru (000) şi alb (111) vârfurilor cubului cu aceleaşi coordonate şi prin validarea umbrii (`glShadeModel (GL_SMOOTH)`).

■ Exemplul 7.2

Pentru generarea unei imagini cu iluminare şi umbrire (cum este cea din fig. 7.9), trebuie să se valideze sistemul de iluminare şi să se definească cel puțin o sursă de lumină şi parametrii materialului fețelor obiectelor. Programul care generează imaginea din fig. 7.9, scris folosind sistemul de dezvoltare GLUT este următorul:

```
#include <GL/glut.h>
void Init(){
   GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
   GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
   GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
   GLfloat mat_ambient[] = {0.1, 0.1, 0.1, 1.0};
   GLfloat mat_diffuse[] = {0.9, 0.9, 0.9, 1.0};
   GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
   GLfloat mat_shininess =  50.0 ;
   glClearColor(0.0,0.0,0.0,0.0);
```

```
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
    glShadeModel(GL_SMOOTH);                    // implicit
}
void Display(void){
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glPushMatrix();
       glTranslatef(-1.5,0,-8);          //transf. modelare
       glShadeModel(GL_FLAT);
       glutSolidSphere(1,16,16);
    glPopMatrix();
    glPushMatrix();
       glTranslatef(1.5,0,-8);           // transf. modelare
       glShadeModel (GL_SMOOTH);
       glutSolidSphere(1,16,16);
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,(GLfloat)w/(GLfloat)h, 1, 40.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH );
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Lights");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

Funcţiile callback `Display()` şi `Reshape()` au aceleaşi semificaţii descrise în programele precedente. Sursa de lumină direcţională (plasată la distanţă infinită) are culoarea albă, iar materialul are componenta specular albă, iar componentele difuză şi ambientală au nuanţe de gri. Pentru redarea primei sfere s-a setat umbrire poligonală (`GL_FLAT`), iar pentru cea de-a doua sferă umbrire Gouraud (`GL_SMOOTH`). Imaginea din fig. 7.9 evidenţiază diferenţa dintre umbrirea poligonală şi umbrirea Gouraud, precum şi iluminarea accentuată (pata luminoasă) datorată reflexiei speculare.

Dacă în programul de mai sus se înlocuieşte componenta de difuzie a materialului cu o culoare roşie prin instrucţiunea:

```
GLfloat mat_diffuse[] = {0.9, 0.1, 0.2, 1.0};
```

se obţine imaginea colorată din Planşa 4, în care este mai evidentă deosebirea dintre umbrirea poligonală şi umbrirea Gouraud.

Aspectul realist al materialelor din care sunt construite obiectele se obţine prin combinarea proprietăţilor de reflexie difuză, specular şi ambientală ale materialului. În Planşa 3 sunt reprezentate 24 de obiecte ceainic, care par a fi din diferite materiale reale.

7.5.3 NORMALELE ÎN VÂRFURILE PRIMITIVELOR GEOMETRICE

Pentru calculul reflexiei difuze şi speculare este necesar să fie definite normalele în vârfurile primitivelor geometrice. Normala care se foloseşte pentru calculul reflexiei într-un vârf este normala curentă, care se setează prin apelul unei variante a funcţiei `glNormal#()` şi se memorează într-o variabilă de stare a bibliotecii, identificată prin constanta simbolică `GL_CURRENT_NORMAL`. O parte din prototipurile funcţiilor `glNormal#()` definite în fişierul header gl.h sunt:

```
void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);
void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz);
void glNormal3i(GLint nx, GLint ny, GLint nz);
void glNormal3dv(const GLdouble *v);
void glNormal3fv(const GLfloat *v);
```

Normala se poate specifica prin componentele `nx`, `ny`, `nz`, sau printr-un pointer la un vector care conţine cele trei componente, de tipul cerut prin numele funcţiei `glNormal#()`.

Normalele se poate defini în orice loc în program, în exteriorul sau în interiorul unui bloc `glBegin()-glEnd()`. Dacă normala se defineşte o singură dată pentru o primitivă geometrică, această valoare este folosită pentru calculul intensităţilor tuturor vârfurilor primitivei geometrice. În mod normal însă, pentru calculul corect al intensităţilor culorii în vârfuri, este necesar ca normala să fie definită pentru fiecare vârf al primitivei geometrice, deci funcţiile `glNormal#()` se apelează alternant cu funcţiile `glVertex#()` în interiorul blocului `glBegin()-glEnd()`.

În exemplul precedent nu a fost necesară definirea normalelor, deoarece funcţia `glutSolidSphere()` le defineşte în mod automat, dar, pentru calculul iluminării unui obiect definit printr-o mulţime de primitive date prin vârfurile lor, este necesar să fie definite toate normalele în vârfuri.

Normalele se specifică în acelaşi sistem de referinţă ca şi vârfurile primitivelor geometrice, deci în sistemul de referinţă de modelare. Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, şi memorate ca parte a modelului obiectului în baza de date grafice. Asupra lor se aplică în mod automat transformarea de modelare-vizualizare folosind valoarea matricei din vârful stivei de modelare-vizualizare, existentă în momentul apelului funcţiei `glNormal#()`. În felul acesta, normalele sunt transformate în sistemul de referinţă de observare, unde se calculează intensităţile componentelor de reflexie în vârfurile primitivelor geometrice.

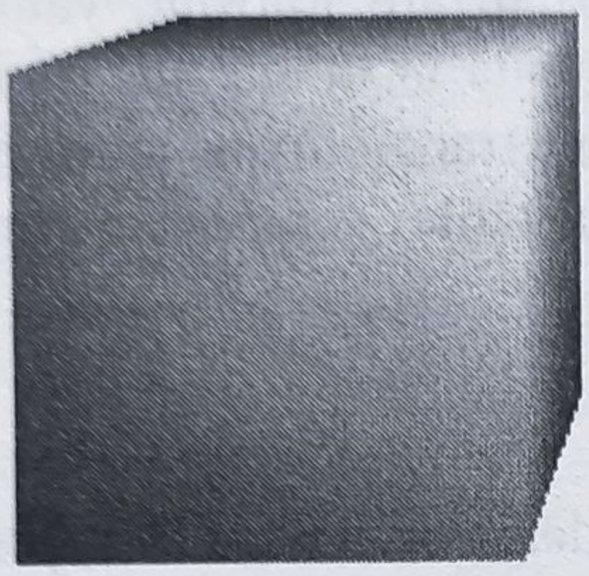
■ **Exemplul 7.3**

În acest exemplu se evidenţiază necesitatea definirii normalelor pentru calculul iluminării. În program este validat sistemul de iluminare şi este definită o sursă de lumină direcţională şi materialul, cu componente de reflectanţă difuză şi ambientală. Cubul din fig. 7.13(a) a fost generat cu definirea normalelor în fiecare vârf al fiecărei suprafeţe. În fig. 7.13 (b), unde singura diferenţă este absenţa normalelor în vârfuri; se observă absenţa umbririi obiectului.

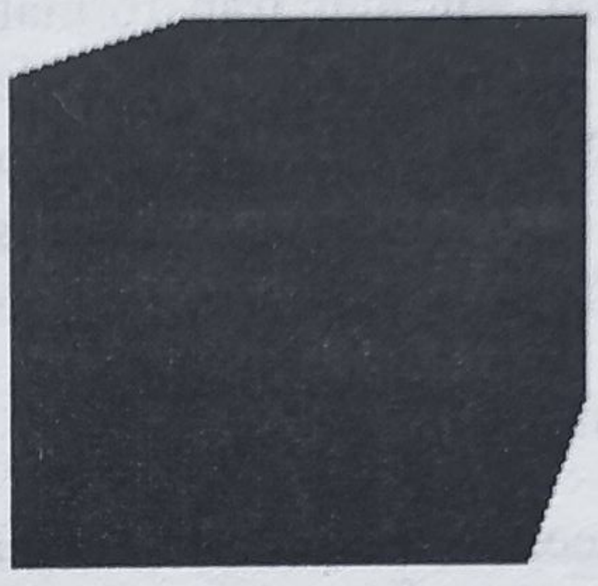
Programul care produce aceste imagini este foarte asemănător cu programul din exemplul 7.2 pentru toate funcţiile, mai puţin funcţia `Display()`, care arată astfel:

```
void Display(){
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glEnable (GL_NORMALIZE);
   glPushMatrix();
      glTranslated(-2,-2,-8);  // transf. de observare
      glLightfv(GL_LIGHT0,
                GL_POSITION, light_position);
   glBegin(GL_POLYGON);
      glNormal3d(-1,1, 1);
      glVertex3d(-1,1, 1);
      glNormal3d( 1,1, 1);
      glVertex3d( 1,1, 1);
      glNormal3d( 1,1,-1);
      glVertex3d( 1,1,-1);
      glNormal3d(-1,1,-1);
      glVertex3d(-1,1,-1);
   glEnd();
   glBegin(GL_POLYGON);
      glNormal3d(-1,-1,1);
      glVertex3d(-1,-1,1);
      glNormal3d( 1,-1,1);
      glVertex3d( 1,-1,1);
```

```
            glNormal3d( 1, 1,1);
            glVertex3d( 1, 1,1);
            glNormal3d(-1, 1,1);
            glVertex3d(-1, 1,1);
        glEnd();
        ..........................................
        glPopMatrix();
        glutSwapBuffers();
    }
```



(a)



(b)

Fig. 7.13 (a) Calculul iluminării folosind normalele în vârfuri.
(b) Imaginea obținută dacă nu sunt definite normalele în vârfuri.

În funcția `Display()` sunt introduse numai două din cele şase fețe ale cubului; celelate trei se definesc în mod similar. Funcția `glEnable (GL_NORMALIZE)` are ca efect scalarea la valoarea unitate a vectorilor normalelor introduse prin funcțiile `glNormal#()`.

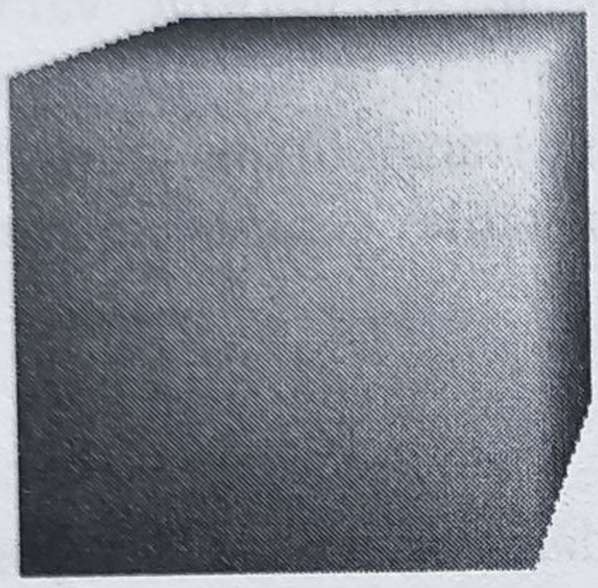
Dacă se comentează toate funcțiile `glNormal3d()` din programul anterior, se obține imaginea din fig. 7.13(b), în care nu se calculează umbrirea fețelor cubului.

7.5.4 CONTROLUL POZIȚIEI ŞI AL DIRECȚIEI SURSELOR DE LUMINĂ

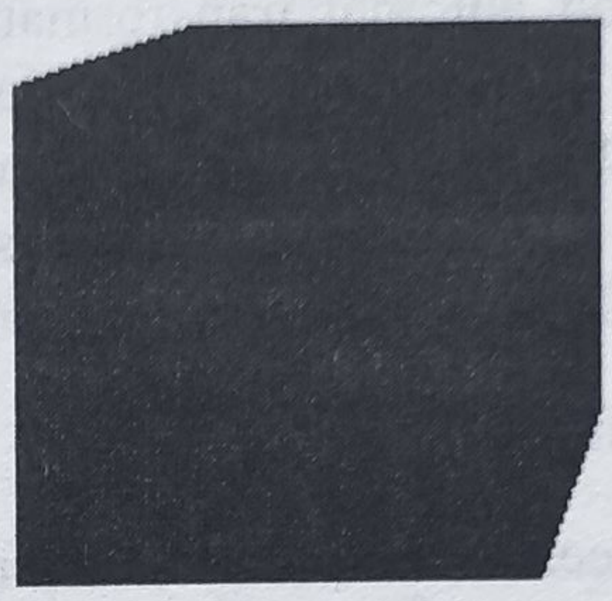
OpenGL tratează poziția şi direcția surselor de lumină la fel cum este tratată poziția primitivelor geometrice, adică sursele de lumină sunt transformate cu aceleaşi matrice de transformare ca şi primitivele geometrice. Atunci când este apelată funcția `glLight#(GL_LIGHTi, GL_POSITION vect)` pentru specificarea poziției sau direcției unei surse de lumină, poziția sau direcția este transformată de matricea de modelare-vizualizare curentă şi este memorată în coordonate în sistemul de referință de observare. Acest lucru înseamnă că se poate modifica poziția sau direcția unei surse de lumină prin modificarea corespunzătoare a matricei de modelare-vizualizare. Matricea de proiecție nu modifică poziția surselor de lunimă. Sursele de lumină pot fi tratate în trei moduri:

- surse cu poziție fixă;
- surse care se deplasează odată cu punctul de observare;
- surse care se deplasează în scenă.

```
            glNormal3d( 1, 1,1);
            glVertex3d( 1, 1,1);
            glNormal3d(-1, 1,1);
            glVertex3d(-1, 1,1);
        glEnd();
        ..........................................
        glPopMatrix();
        glutSwapBuffers();
}
```



(a)



(b)

Fig. 7.13 (a) Calculul iluminării folosind normalele în vârfuri.
(b) Imaginea obținută dacă nu sunt definite normalele în vârfuri.

În funcția `Display()` sunt introduse numai două din cele şase fețe ale cubului; celelate trei se definesc în mod similar. Funcția `glEnable (GL_NORMALIZE)` are ca efect scalarea la valoarea unitate a vectorilor normalelor introduse prin funcțiile `glNormal#()`.

Dacă se comentează toate funcțiile `glNormal3d()` din programul anterior, se obține imaginea din fig. 7.13(b), în care nu se calculează umbrirea fețelor cubului.

7.5.4 CONTROLUL POZIȚIEI ŞI AL DIRECȚIEI SURSELOR DE LUMINĂ

OpenGL tratează poziția şi direcția surselor de lumină la fel cum este tratată poziția primitivelor geometrice, adică sursele de lumină sunt transformate cu aceleaşi matrice de transformare ca şi primitivele geometrice. Atunci când este apelată funcția `glLight#(GL_LIGHTi, GL_POSITION vect)` pentru specificarea poziției sau direcției unei surse de lumină, poziția sau direcția este transformată de matricea de modelare-vizualizare curentă şi este memorată în coordonate în sistemul de referință de observare. Acest lucru înseamnă că se poate modifica poziția sau direcția unei surse de lumină prin modificarea corespunzătoare a matricei de modelare-vizualizare. Matricea de proiecție nu modifică poziția surselor de lunimă. Sursele de lumină pot fi tratate în trei moduri:

- surse cu poziție fixă;
- surse care se deplasează odată cu punctul de observare;
- surse care se deplasează în scenă.

Surse de lumină cu poziţie fixă. Funcţia `glLight#()` cu parametrul `GL_POSITION` defineşte coordonatele omogene ale poziţiei sursei. Dacă această funcţie este apelată în momentul în care matricea curentă din stiva de modelare-vizualizare conţine matricea de observare, atunci poziţia sursei este transformată în coordonate de observare. Poziţia în spaţiu a sursei rămîne fixă, iar poziţia calculată şi utilizată în calcule este poziţia în coordonate de observare. În exemplul 7.3 sursa de lumină este menţinută în poziţie fixă în scenă prin setarea poziţiei după introducerea transformării de observare:

```
glPushMatrix();
   glTranslated(-2,-2,-8);  // transf. de observare
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   // Desenare obiect
glPopMatrix();
```

■ Exemplul 7.4

Se reia programul de desenare a două sfere luminate din exemplul 7.2. Culoarea de ştergere este modificată (culoare gri deschis (0.8, 0.8, 0.8)) pentru o mai bună evidenţiere a umbririi obiectelor. Transformările de instanţiere se aplică fiecărui obiect în parte. Transformarea de observare se aplică tuturor obiectelor şi direcţiei sursei de lumină. În continuare este prezentată numai funcţia `Display()` pentru evidenţierea poziţiei fixe a sursei de lumină, în condiţiile modificării punctului de observare ($x_V = 0$, $y_V = 0$, $z_V = 8$ şi unghiul χ varibil).

```
void Display(){
   GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
      glTranslatef(0,0,-8);
      glRotated(gamma,0,1,0);        // transf. observare
      glLightfv(GL_LIGHT0,
                GL_POSITION, light_position);
      glPushMatrix();
         glTranslatef(1.5,0,0);      // transf. modelare
         glutSolidSphere(1,16,16);
      glPopMatrix();
      glPushMatrix();
         glTranslatef(-1.5,0,0);     // transf. modelare
         glutSolidSphere(1,16,16);
      glPopMatrix();
   glPopMatrix();
   glutSwapBuffers();
}
```

Prin mişcarea de rotaţie a punctului de observare, sferele sunt văzute iluminate în diferite situaţii: pentru $\chi = 45°$ se poate observa partea cea mai luminată a sferelor; pentru $\chi = 225°$ se observă partea cea mai puţin luminată a sferelor (fig. 7.14).

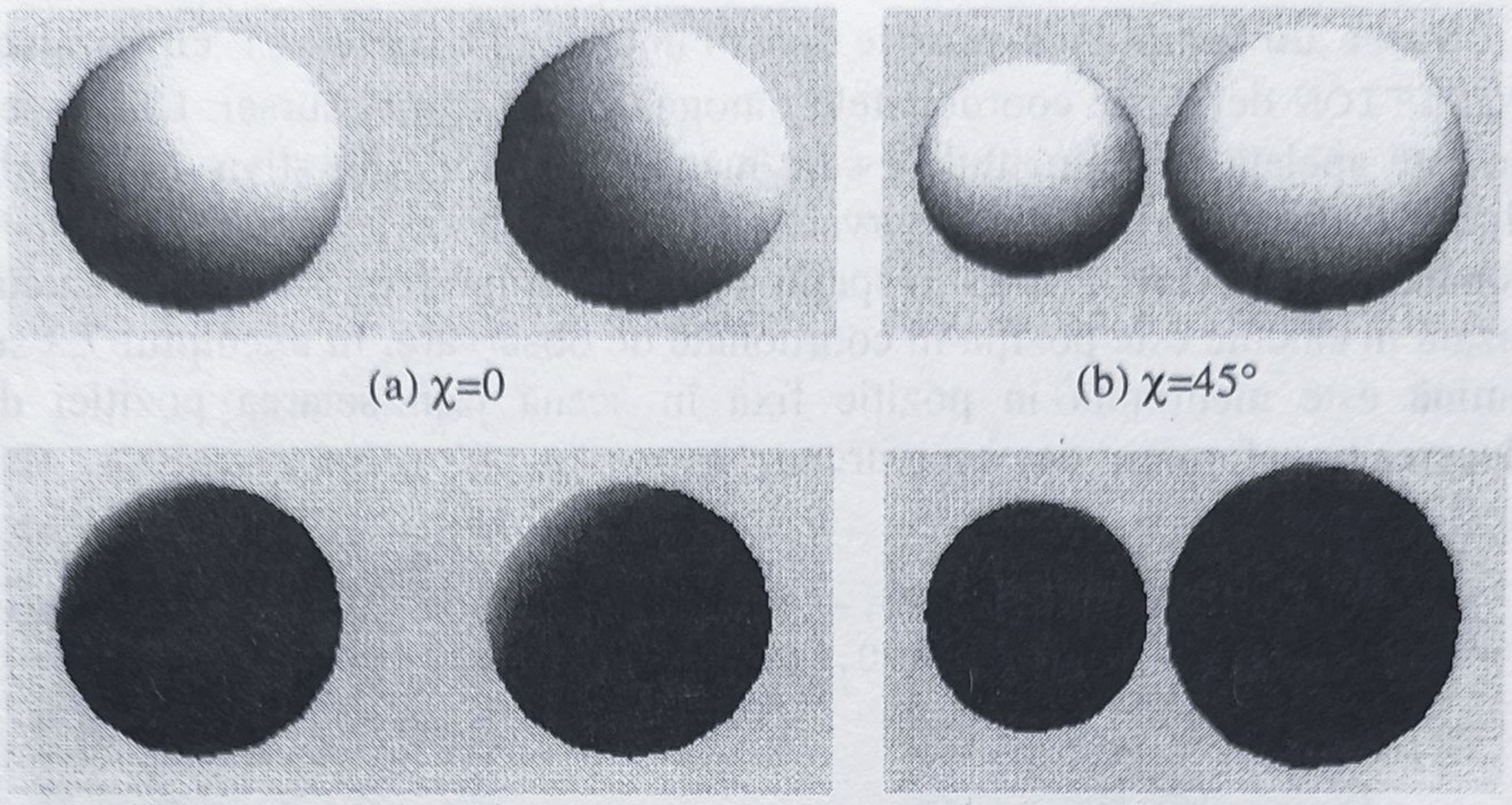


Fig. 7.14 Sursă de lumină fixă în direcția (1.0, 1.0, 1.0, 0.0).

Surse de lumină care se deplasează odată cu punctul de observare. Pentru a crea surse de lumină care se mişcă odată cu punctul de observare, trebuie ca poziția sursei să fie setată înainte de transformarea de observare. Poziția sau direcția sursei date prin funcția `glLight#()` cu parametrul `GL_POSITION` este transformată cu matricea curentă din stiva de modelare-vizualizare, care conține matricea identitate, deci localizarea sursei rămâne constantă față de punctul de observare. Acest mod de control al sursei este folosit în simulatoare de antrenament, pentru a simula lumina farurilor vehiculului. Dacă poziția sursei este (0,0,0,1), atunci sursa este plasată chiar în punctul de observare.

■ Exemplul 7.5

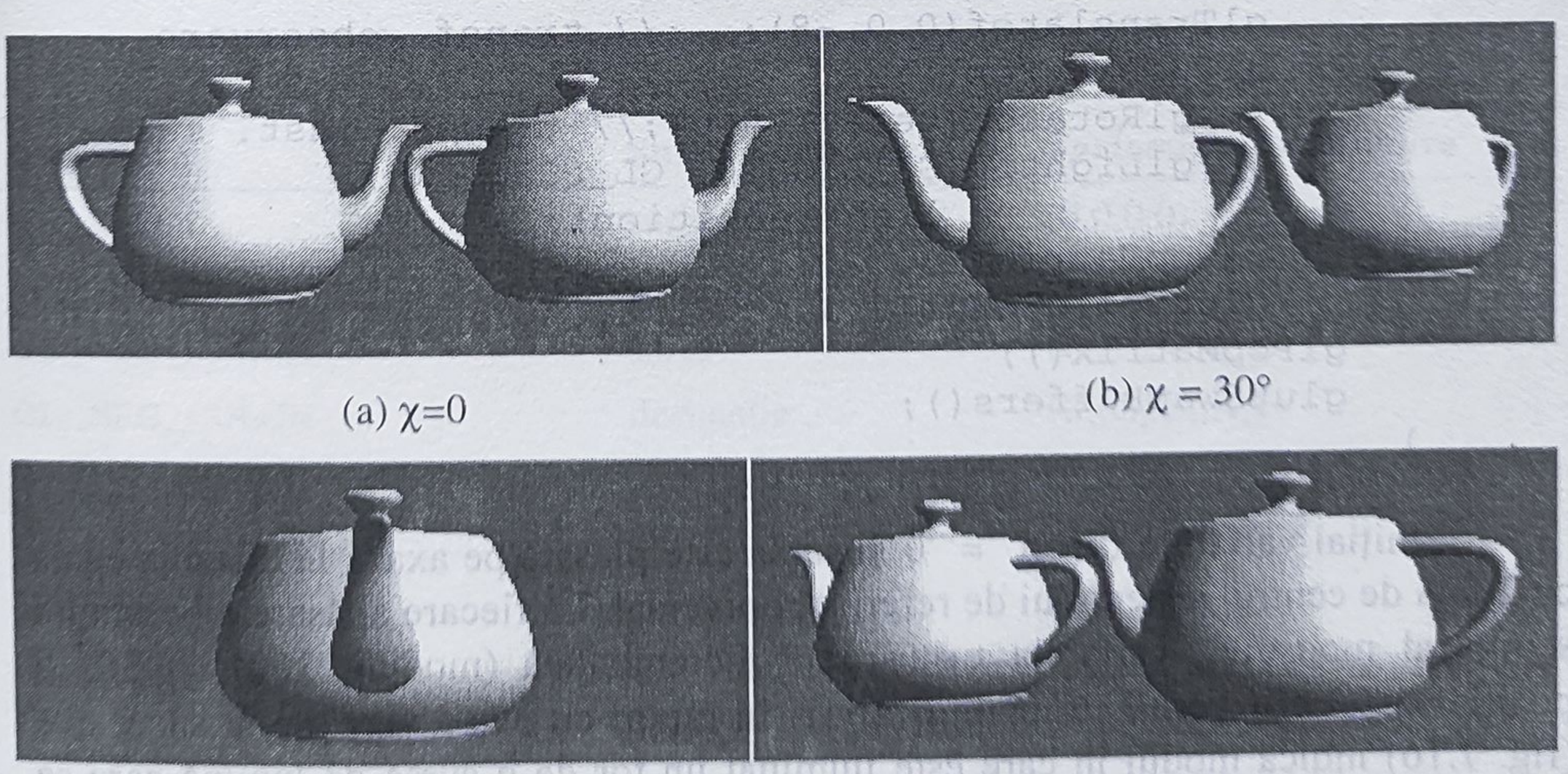
Se reia programul de generare a imaginii obiectelor luminate din exemplul precedent, dar sursa de lumină se menține cu o direcție fixă în sistemul de referință de observare, prin setarea direcției sursei înainte de transformarea de observare. În continuare sunt prezentate funcțiile callback `Display()` şi `Keyboard()` ale programului, celelalte funcții fiind nemodificate (cu excepția culorii de ştergere care este gri închis) şi a înregistrării funcției callback `Keyboard()`).

```
static int gamma = 0;
void Keyboard(unsigned char key, int x, int y){
   switch (key){
   case 'G':
      gamma = (gamma+30)%360;
      glutPostRedisplay();
      break;
   case 'g':
      gamma = (gamma-30)%360;
```

```
            glutPostRedisplay(); break;
        }
    }
void Display(){
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glPushMatrix();
        glTranslatef(0,0,-8);
        glRotated(gamma,0,1,0);          // transf. observare
        glPushMatrix();
            glTranslatef(1.5,0,0);       // transf. modelare
            glutSolidTeapot(1);
        glPopMatrix();
        glPushMatrix();
            glTranslatef(-1.5,0,0);      // transf. modelare
            glutSolidTeapot(1);
        glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();
}
```

Sursa de lumină direcțională are direcția (1.0, 1.0, 1.0, 1.0) care, transformată cu matricea identitate, păstrează aceeaşi valoare în sistemul de referință de observare. De aceea, indiferent de poziția şi direcția de observare, obiectele sunt iluminate dintr-o direcție fixă față de direcția de observare.

În fig. 7.15 sunt reprezentate mai multe imagini, pentru direcții de observare diferite. Unghiul χ (gamma în program) de rotație față de axa y se modifică prin comenzi de la tastatură, prelucrate de funcția callback `Keyboard()`.



(a) χ=0 (b) χ = 30°

(c) χ=90° (c) χ=150°

Fig. 7.15 Sursă de lumină care se deplasează odată cu punctul de observare.

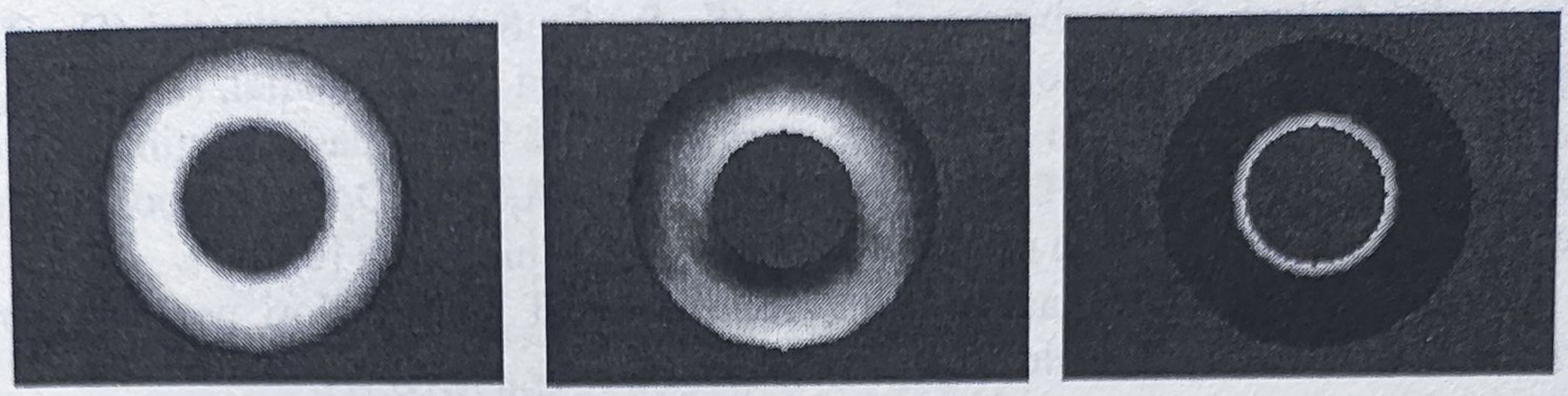
Surse de lumină care se deplasează în scenă. Se pot defini surse de lumină care se mişcă independent în scena virtuală, prin aplicarea unei transformări de instanţiere (care localizează sursa de lumină în sistemul de referinţă universal), urmată de transformarea de observare, prin care se calculează poziţia sau direcţia sursei în sistemul de referinţă de observare. La fel ca şi în cazul instanţierii obiectelor în scenă, se foloseşte matricea curentă din stiva de modelare-vizualizare.

■ Exemplul 7.6

Se consideră un punct şi o direcţie de observare fixă, iar obiectele din scenă sunt luminate de o sursă de lumină direcţională, care îşi modifică direcţia printr-o comandă de la mouse. Programul care implementează această funcţionare este similar celorlalte programe de iluminare prezentate în exemplele precedente, cu excepţia funcţiei `Display()` şi a funcţiei callback de prelucrare a evenimentelor de la mouse, care trebuie înregistrată în GLUT.

```
#include <GL/glut.h>
static int teta = 0;
void Mouse(int button, int state, int x, int y){
   switch(button){
   case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN){
         teta = (teta+30)%360;
         glutPostRedisplay();}
   }
}
void Display(){
   GLfloat light_position[] = {0, 0, 2, 1.0};
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
      glTranslatef(0,0,-8);    // transf. observare
      glPushMatrix();
         glRotated(teta,1,0,0);// transf. inst. sursa
         glLightfv(GL_LIGHT0, GL_POSITION,
                   light_position);
      glPopMatrix();
      glutSolidTorus(0.3, 0.8, 8, 16);
   glPopMatrix();
   glutSwapBuffers();
}
```

Iniţial variabila `teta = 0` şi sursa este plasată pe axa `z` la distanţa egală cu 2 faţă de centrul sistemului de referinţă universal. La fiecare apăsare a butonului stânga al mousului, variabila `teta` este incrementată (modulo 360) cu 30, şi poziţia sursei este rotită cu unghiul `spin` în raport cu axa x. Imaginile succesive (fig. 7.16) indică modul în care este iluminat un tor de o sursă de lumină care se roteşte în jurul axei x.



(a) $\theta = 0°$ (b) $\theta = 60°$ (c) $\theta = 180°$

Fig. 7.16 Sursă de lumină care se deplasează în scenă.

7.5.5 COMBINAREA CULORILOR

Mai multe efecte în redarea obiectelor (transparență, anti-aliasing, ceață) se obțin prin combinarea culorilor la nivel de pixel.

Pentru validarea operației de combinare a culorilor la nivel de pixel (*blending*) se apelează funcția `glEnable(GL_BLEND)`. În cursul combinării, culoarea unui nou fragment (sursa) este combinată cu culoarea memorată într-o locație corespunzătoare din bufferul de cadru, care este şi destinația pentru înscrierea rezultatului combinării.

Pentru combinarea culorilor se calculează factorii de combinare ai sursei şi ai destinației. Aceşti factori sunt cvadrupleți RGBA, care se multiplică cu componentele corespunzătoare R, G, B şi A ale sursei, respectiv ale destinației. În tabelul 7.2 sunt date valorile constantelor simbolice predefinite în OpenGL, din care se pot selecta factorii de combinare ai sursei şi ai destinației.

Tabelul 7.2

Factorii de combinare ai sursei şi ai destinației

Constanta	Sursă-destinație	Factorul de combinare
GL_ZERO	sursă sau dest	(0,0,0,0)
GL_ONE	sursă sau dest	(1,1,1,1)
GL_DST_COLOR	sursă	(R_d,G_d,B_d,A_d)
GL_SRC_COLOR	destinație	(R_s,G_s,B_s,A_s)
GL_ONE_MINUS_DST_COLOR	sursă	$(1,1,1,1) - (R_d,G_d,B_d,A_d)$
GL_ONE_MINUS_SRC_COLOR	destinație	$(1,1,1,1) - (R_s,G_s,B_s,A_s)$
GL_SRC_ALPHA	sursă sau dest	(A_s,A_s,A_s,A_s)
GL_DST_ALPHA	sursă sau dest	(A_d,A_d,A_d,A_d)
GL_ONE_MINUS_SRC_ALPHA	sursă sau dest	$(1,1,1,1) - (A_s,A_s,A_s,A_s)$
GL_ONE_MINUS_DST_ALPHA	sursă sau dest	$(1,1,1,1) - (A_d,A_d,A_d,A_d)$
GL_SRC_ALPHA_SATURATE	sursă	$(f,f,f,1)$; $f = \min(A_s, 1-A_d)$

Fie factorii de combinare ai sursei (S_r, S_g, S_b, S_a) şi ai destinaţiei (D_r, D_g, D_b, D_a) şi componentele culorii sursei (R_s, G_s, B_s, A_s) şi ale destinaţiei, (R_d, G_d, B_d, A_d). Culoarea rezultată prin combinare, care se înscrie în locaţia de destinaţie, are componentele ($R_sS_r+R_dD_r$, $G_sS_g+G_dD_g$, $B_sS_b+G_dD_g$, $A_sS_a+A_dD_a$). Este posibil ca fiecare componentă a culorii rezultate să fie limitată la intervalul [0,1].

Factorii de combinare se pot selecta din mai multe valori posibile prin argumentele transmise funcţiei:

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
```

7.5.5.1 Transparenţa

Transparenţa suprafeţelor se implementează folosind combinarea cu factorii de combinare `GL_SRC_ALPHA`, pentru sursă şi `GL_ONE_MINUS_SRC_ALPHA` pentru destinaţie.

■ **Exemplul 7.7**

În programul următor se suprapun parţial suprafeţe transparente triunghiulare de culoare gri deschis (0.8, 0.8, 0.8) peste un dreptunghi de culoare gri închis (0.2, 0.2, 0.2).

În continuare sunt prezentate funcţiile callback `Init()` şi `Display()` ale programului dezvoltat sub GLUT. Celelalte funcţii, `main()` şi `Reshape()` nu au nimic special, fiind asemănătoare cu cele din programele precedente.

```
void Init(){
   glClearColor(1.0,1.0,1.0,1.0);
   glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}
void Triangle() {
   glBegin(GL_POLYGON);
      glVertex3d(-2,-2,0);
      glVertex3d( 2,-2,0);
      glVertex3d( 0, 2,0);
   glEnd();
}
void Display(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
   glTranslated(0.0,0.0,-20.0);// transf. de observare
   glDisable(GL_BLEND);
   glColor3d(0.2,0.2,0.2);
   glBegin(GL_POLYGON);          // desenare dreptunghi
      glVertex3d(-8,0,0);
      glVertex3d( 8,0,0);
      glVertex3d( 8,4,0);
      glVertex3d(-8,4,0);
   glEnd();
   glEnable (GL_BLEND);
```

```
    glPushMatrix();
       glTranslated(-6.0,4.0,0.0);
       glColor4d(0.8,0.8,0.8,0.25);
       Triangle();                    // primul triunghi
    glPopMatrix();
    glPushMatrix();
       glTranslated(-2.0,4.0,0.0);
       glColor4d(0.8,0.8,0.8,0.50);
       Triangle();                    // al doilea triunghi
    glPopMatrix();
    glPushMatrix();
       glTranslated(2.0,4.0,0.0);
       glColor4d(0.8,0.8,0.8,0.75);
       Triangle();                    // al treilea triunghi
    glPopMatrix();
    glPushMatrix();
       glTranslated(6.0,4.0,0.0);
       glColor4d(0.8,0.8,0.8,1.0);
       Triangle();                    // al patrulea triunghi
    glPopMatrix();

    glPopMatrix();
    glutSwapBuffers();
}
```

În funcția `Init()` se setează culoarea de ştergere albă (1.0, 1.0, 1.0, 1.0) a bufferului de culoare şi factorii de combinare `GL_SRC_ALPHA`, pentru sursă şi `GL_ONE_MINUS_SRC_ALPHA` pentru destinație.

Imaginea din fig. 7.16 s-a obținut pentru valorile 0.25, 0.50, 0.75, 1.0 ale componentei A_s a triunghiului. Combinarea culorii unei noi suprafețe cu culoarea existentă în buffer produce aspectul de transparență a suprafețelor.

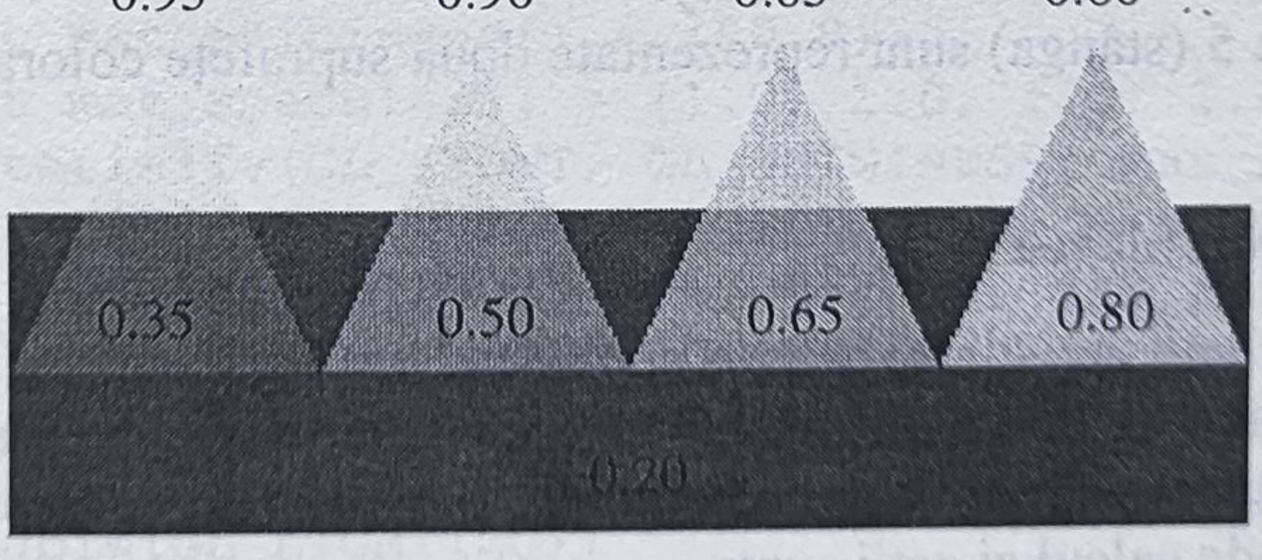


Fig. 7.16 Combinarea culorilor în calculul transparenței.

Dreptunghiul se înscrie în buffer cu culorea lui inițială. Culoarea fiecăruia din cele 4 triunghiuri care urmează (poligoane sursă) se combină la nivel de pixel cu culoarea existentă în buffer (care poate fi culoarea de ştergere sau culoarea dreptunghiului) folosind pentru factorii de combinare valoarea transparenței sale.

Primele trei triunghiuri sunt compuse din două culori: una rezultată din combinarea culorii proprii cu culoarea dreptunghiului, cealaltă rezultată din combinarea culorii proprii cu culoarea de ştergere. De exemplu, pentru primul triunghi, în partea de jos, suprapusă peste dreptunghi, fiecare componentă de culoare R,G,B are intensitatea 0.25*0.8 + (1 – 0.25)*0.2 = 0.35; în partea de sus, prin combinare cu culoarea de ştergere, fiecare componentă de culoare R,G,B rezultată are intensitatea 0.25*0.8 + (1 – 0.25)*1.8 = 0.95. Pentru al doilea triunghi componentele de culoare sunt 0.5 pentru partea de jos şi 0.9 pentru partea de sus; pentru al treilea triunghi componentele de culoare sunt 0.65 pentru partea de jos şi 0.85 pentru partea de sus. Ultimul triunghi, cu $A_s = 1$, înlocuieşte complet valorile existente în buffer şi este desenat cu componentele de culoare 0.8 în ambele părți de suprapunere.

De remarcat că în acest program simplu nu este validat bufferul de adâncime. În mod implicit, în OpenGL testul de adâncime este invalidat şi, deoarece nu se validează nici la inițializare nici în altă parte în program, testul de adâncime nu se efectuează şi poligoanele sunt înscrise în bufferul de imagine în ordinea în care sunt transmise, indiferent de adâncimea lor.

Acest fapt trebuie să fie evitat atunci când sunt reprezentate obiecte tridimensionale opace şi transparente în scenă. Dat fiind că obiectele tridimensionale se pot ascunde unele pe altele este necesar să se valideze testul de adâncime prin apelul funcției `glEnable(GL_DEPTH_TEST)`. Obiectele opace se desenează mai întâi şi se execută operațiile cu bufferul de adâncime în mod normal. Pentru desenarea obiectelor transparente, se setează condiția de acces numai pentru citire a bufferului de adâncime (read-only) prin apelul funcției `glDepthMask(GL_FALSE)`. Atunci când se desenează obiecte transparente, adâncimea lor se compară cu adâncimile memorate în Z-buffer, stabilite de obiectele opace, astfel că nu se desenează părțile ascunse (aflate în spatele) obiectelor opace. Dacă sunt mai aproape de punctul de observare, se desenează prin combinarea culorii, dar nu se elimină adâncimea obiectele opace din bufferul de adâncime, deoarece acesta este read-only.

În Planşa 5 (stânga) sunt reprezentate două suprafețe colorate suprapuse cu transparență.

7.5.5.2 Simularea ceții

Calculul ceții se validează prin apelul funcției `glEnable(GL_FOG)`. Se pot defini trei moduri de calcul af funcției de combinare *f*, corespunzător unei variații exponențiale, pătratic exponențiale şi liniare. Funcțiile prin care se pot defini parametrii de calcul ai ceții sunt:

```
void glFogf(GLenum pname, GLfloat param);
void glFogi(GLenum pname, GLint param);
void glFogfv(GLenum pname, const GLfloat *params);
void glFogiv(GLenum pname, const GLint *params);
```

Parametrul `pname` poate fi una din valorile `GL_FOG_DENSITY`, `GL_FOG_MODE`, `GL_FOG_START`, `GL_FOG_END` pentru definirea modului de

calcul, a densităţii, a distanţei de început şi a distanţei de sfârşit a ceţii. Dacă se defineşte modul de calcul, atunci parametrul `param` poate lua una din constantele simbolice `GL_LINEAR`, `GL_EXP`, `GL_EXP2` . Dacă se defineşte densitatea, atunci `param` va avea o valoare pozitivă care este folosită în calculele exponenţiale. Distanţa de început sau de sfârşit a ceţii, folosită în funcţia liniară de calcul, se transmite prin argumentul `param` corespunzător parametrului `GL_FOG_START`, respectiv `GL_FOG_END`.

Pentru a defini culoarea ceţii se apelează una din funcţiile `glFogiv()` sau `glFogfv()` în care argumentul `pname` are valoarea `GL_FOG_COLOR`, iar argumentul `params` este un pointer la un vector de patru valori întregi sau numere în virgulă flotantă, care conţin componentele R,G,B,A de culoare ale ceţii.

■ Exemplul 7.8

În acest exemplu se desenează cinci obiecte la distanţe diferite. Dat fiind că se validează calculul ceţii, obiectele se văd din ce în ce mai estompate, culoarea lor apropiindu-se de culoarea de ştergere cu care se face combinarea.

```
#include <GL/glut.h>
static GLuint listName;
void Teapot (GLfloat x, GLfloat y, GLfloat z){
    glPushMatrix();
    glTranslatef (x, y, z);
    glutSolidTeapot(0.9);
    glPopMatrix();
}
void Init(){
   float fogDensity = 0.04;
   float fog_color[] = {0.5, 0.5, 0.5, 1.0};
   float mat_ambient[] = {0.2, 0.0, 0.0, 1.0};
   float mat_diffuse[] = {0.6, 0.0, 0.0, 1.0};
   float mat_shininess[] = {90.0};
   float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glEnable(GL_FOG);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glFogi(GL_FOG_MODE, GL_EXP);
   glFogf(GL_FOG_DENSITY, fogDensity);
   glFogfv(GL_FOG_COLOR, fog_color);
   glClearColor(0.5, 0.5, 0.5, 1.0);
   listName = glGenLists(1);
   glNewList(listName,GL_COMPILE);
```

```
        Teapot (-3.0, -0.5, -10.0);
        Teapot (-2.0, -0.5, -20.0);
        Teapot (0.0, -0.5, -30.0);
        Teapot (2.5, -0.5, -40.0);
        Teapot (5.0, -0.5, -50.0);
    glEndList();
}
void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glCallList(listName);
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,(GLfloat)w/(GLfloat)h,0.1,4000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Simularea cetii");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

Programul creează o listă de display care conține apelurile funcțiilor OpenGL pentru generarea imaginii. La execuția acestui program se obține imaginea din Planşa 5 (dreapta).

8

MODELAREA ŞI REDAREA SUPRAFEȚELOR PARAMETRICE

Cele mai utilizate tipuri de curbe şi suprafețe parametrice în proiectarea grafică sunt curbele şi suprafețele Bézier şi B-spline, care vor fi descrise în continuare.

8.1 CURBE BÉZIER

Cea mai cunoscută formă de reprezentare parametrică a curbelor şi suprafețelor este cea dezvoltată de Pierre Bézier pentru proiectarea caroseriei maşinilor Renault [Bez72]. Sistemul de proiectare UNISURF, bazat pe reprezentarea parametrică a suprafețelor, a fost folosit pentru proiectarea multor modele de maşini Renault.

Curbele Bézier sunt curbe parametrice cubice în care se folosesc patru puncte de control, P_0, P_1, P_2, P_3, iar funcțiile de bază sunt funcțiile Bézier, care au următoarele expresii:

$$\begin{aligned} B_0(u) &= (1-u)^3 \\ B_1(u) &= 3u(1-u)^2 \\ B_2(u) &= 3u^2(1-u) \\ B_3(u) &= 3u^3 \end{aligned} \tag{8.1}$$

Rezultă expresia polinomială a curbelor Bézier:

$$\mathbf{Q}(u) = \mathbf{P}_0(1-u)^3 + 3\,\mathbf{P}_1u(1-u)^2 + 3\,\mathbf{P}_2u(1-u) + 3\,\mathbf{P}_3u^3 \tag{8.2}$$

Punctele P_0, P_1, P_2, P_3 se numesc puncte de control deoarece poziția lor în spațiu influențează forma curbei. P_0 şi P_3 sunt capetele curbei. Poligonul obținut prin unirea punctelor succesive se numeşte *poligon caracteristic* sau *poligon de control*. În sensul definiției date poligonului în secțiunea precedentă, poligonul caracteristic ar trebui închis cu ultimul segment, de la P_3 la P_0; în general, însă, acest segment al poligonului caracteristic nu se reprezintă. În fig. 8.1 (a) sunt

arătate trei curbe Bézier cubice, cu poligoanele caracteristice. Programul care generează această imagine este descris în exemplul 8.1.

Vectorii tangenți la curba Bézier se obțin prin derivarea funcțiilor de bază în raport cu parametrul u şi au valorile în capete:

$$\mathbf{Q}_u(0) = 3(\mathbf{P}_1 - \mathbf{P}_0)$$
$$\mathbf{Q}_u(1) = 3(\mathbf{P}_2 - \mathbf{P}_3)$$

Din aceste relații rezultă că punctele P_0 şi P_1 se află pe tangenta la curbă în punctul P_0, iar punctele P_2 şi P_3 se află pe tangenta la curbă în punctul P_3.

Reprezentarea grafică a celor patru funcții de bază Bézier cubice (fig. 8.1(b)) arată în ce proporție influențează un punct de control forma curbei Bézier. Punctul de control P_0 are cea mai mare influență la u = 0, unde $B_0 \doteq 1$, iar celelalte funcții, B_1, B_2, B_3, sunt 0. Punctul de control P_3 are cea mai mare influență la u = 1, unde $B_3 = 1$, iar celelate funcții, B_0, B_1, B_2, sunt egale cu zero. Punctele P_1 şi P_2 au cel mai mare efect la $u \doteq 1/3$, respectiv u = 2/3. Se poate remarca faptul că modificarea fiecărui punct de control influențează, cu un efect mai mare sau mai mic, forma întregii curbe Bézier. Acesta este, de fapt, dezavantajul principal al curbelor Bézier: controlul global al formei curbei.

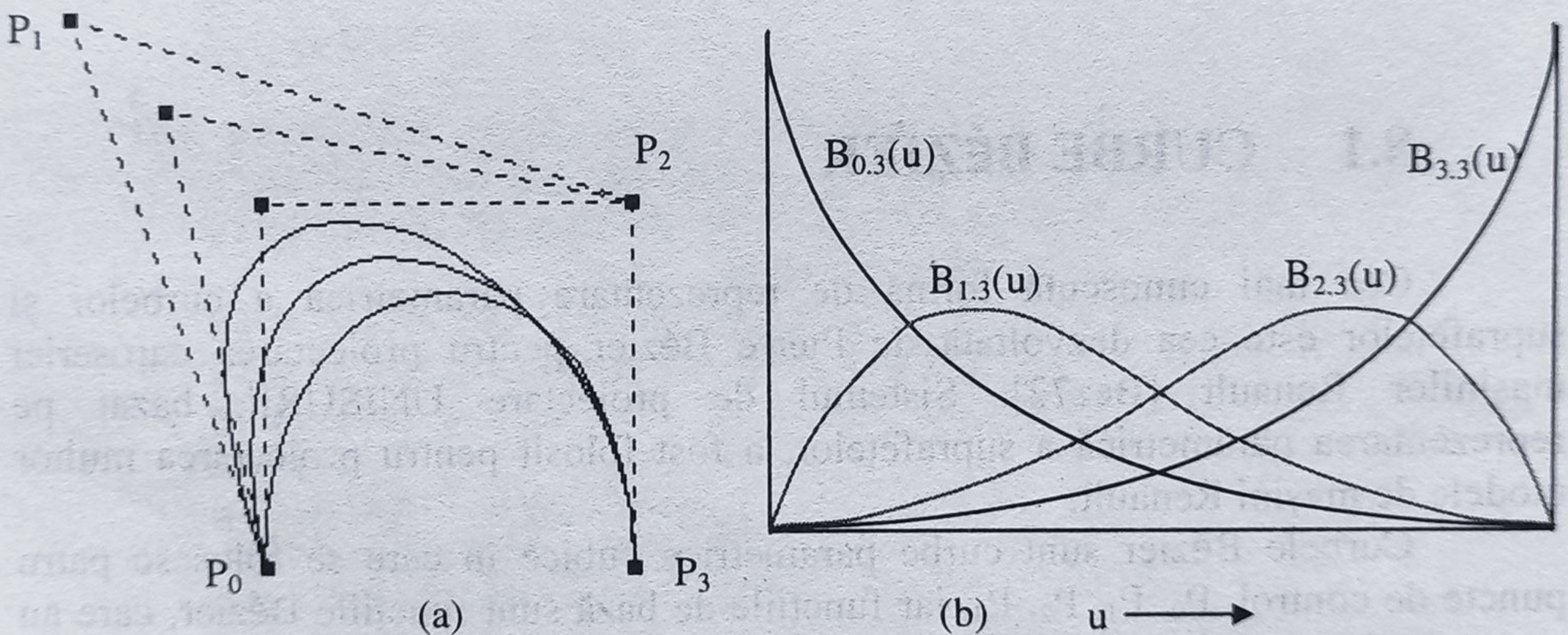


Fig. 8.1 (a) Curbe Bézier cubice şi poligoanele caracteristice.
(b) Funcțiile de bază Bézier cubice.

Funcțiile de bază Bézier cubice exprimate de relația (2.10), pot fi considerate un caz particular al unor funcții de bază de un ordin oarecare n, pentru exprimarea parametrică a unei curbe sub forma:

$$\mathbf{Q}(u) = \sum_{i=0}^{n} \mathbf{P}_i \mathbf{B}_{i,n}(u) \qquad (8.3)$$

unde $\mathbf{B}_{i,n}$ sunt polinoame Bernstein:

$$\mathbf{B}_{i,n}(u) = C(n,i)u^i(1-u)^{n-i} \qquad (8.4)$$

cu coeficienții $C(n,i) = n!/(i!(n-i)!)$.

Pentru funcţii de bază de gradul n sunt necesare n+1 puncte de control; numărul punctelor de control dă ordinul curbei Bézier. Funcţiile de bază de ordin mai mare de trei permit descrierea unor curbe mai complexe, dar timpul de calcul al punctelor de pe curbă este mai ridicat (sunt necesare mai multe operaţii pentru fiecare punct) şi, de aceea, sunt rar utilizate în grafica interactivă sau în modelare. În plus, cu cât creşte gradul funcţiilor de bază, relaţia dintre poligonul caracteristic şi forma curbei devine mai slabă şi, de aceea, proiectantul controlează mai greu forma curbei.

Curbe mai complexe se pot obţine şi prin combinarea mai multor segmente de curbe Bézier cubice. Combinarea mai multor segmente necesită impunerea unor condiţii care să asigure continuitatea segmentelor. Condiţia de continuitate poziţională (de ordinul 0) a segmentelor de curbă este ca punctul de control final al unui segment să fie primul punct de control al segmentului următor (fig.8.2(a)). Această condiţie nu este suficientă, deoarece pot să apară discontinuităţi la redarea obiectelor. Condiţia de continuitate tangenţială (de ordinul 1) este ca segmentele de curbă să aibă aceeaşi tangentă în punctul de alipire. Aceasta înseamnă că este necesar ca muchiile adiacente a celor două poligoane caracteristice să fie coliniare (fig. 8.2(b)).

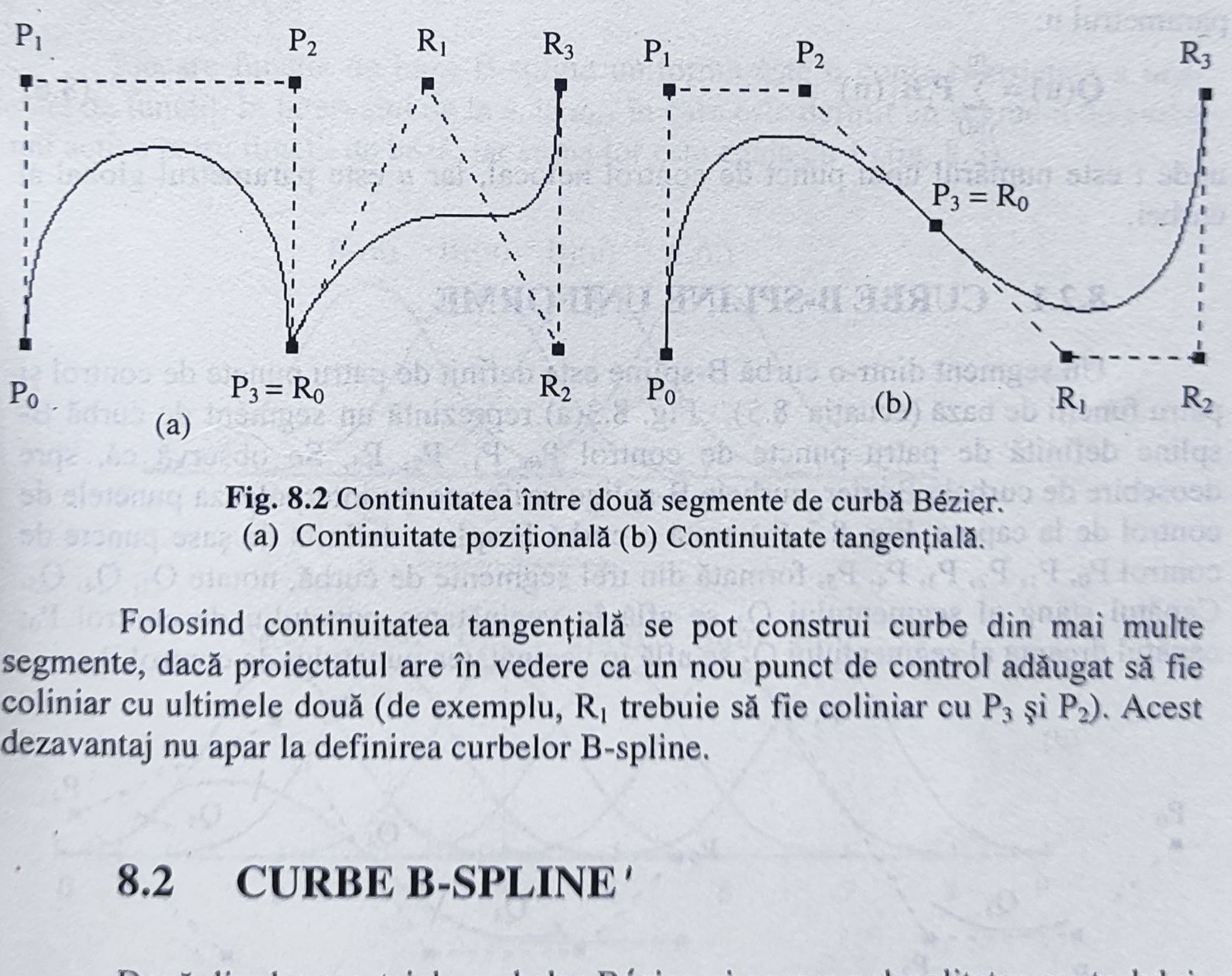


Fig. 8.2 Continuitatea între două segmente de curbă Bézier.
(a) Continuitate poziţională (b) Continuitate tangenţială.

Folosind continuitatea tangenţială se pot construi curbe din mai multe segmente, dacă proiectatul are în vedere ca un nou punct de control adăugat să fie coliniar cu ultimele două (de exemplu, R_1 trebuie să fie coliniar cu P_3 şi P_2). Acest dezavantaj nu apar la definirea curbelor B-spline.

8.2 CURBE B-SPLINE'

Două din dezavantajele curbelor Bézier, şi anume nelocalitatea controlului şi relaţia dintre numărul punctelor de control şi gradul curbei, sunt înlăturate în reprezentarea parametrică a curbelor B-spline.

O curbă B-spline poate fi compusă din oricâte segmente cubice pe porțiuni, cu schimbarea corespunzătoare a coeficienților la trecerea de la un segment la altul. Pentru o secvență de puncte de control P_i, ecuația segmentului $\mathbf{Q}_i(u)$ al curbei B-spline este:

$$\mathbf{Q}_i(u) = \sum_{k=0}^{3} \mathbf{P}_{i-3+k} \mathbf{B}_{i-3+k}(u) \tag{8.5}$$

unde *i* este numărul segmentului de curbă, iar *k* este indexul punctului de control local, adică este indexul pentru segmentul *i*. Folosind această notație se poate considera *u* un parametru local variind în intervalul închis [0,1] pentru definirea segmentului dat.

O curbă B-spline este compusă din $m - 2$ segmente notate $Q_3, Q_4, \ldots Q_m$, controlate de m + 1 puncte de control $P_0, P_1, P_2, \ldots, P_m$, unde $m \geq 3$. Fiecare segment al curbei este definit de patru puncte de control şi fiecare punct de control influențează numai patru segmente din curbă. Aceasta este proprietatea de control local a curbelor B-spline, care este un avantaj față de curbele Bézier.

Un segment de curbă B-spline prezintă continuitate de ordinul 0, 1 şi 2 şi se poate defini o mulțime de segmente de curbă B-spline ca o singură curbă cu parametrul u:

$$\mathbf{Q}(u) = \sum_{i=0}^{m} \mathbf{P}_i \mathbf{B}_i(u) \tag{8.6}$$

unde *i* este numărul unui punct de control nelocal, iar *u* este parametrul global al curbei.

8.2.1 CURBE B-SPLINE UNIFORME

Un segment dintr-o curbă B-spline este definit de patru puncte de control şi patru funcții de bază (ecuația 8.5). Fig. 8.3(a) reprezintă un segment de curbă B-spline definită de patru puncte de control P_0, P_1, P_2, P_3, Se observă că, spre deosebire de curbele Bézier, curbele B-spline uniforme nu interpolează punctele de control de la capete. Fig. 8.3 (b) arată o curbă B-spline definită de şase puncte de control $P_0, P_1, P_2, P_3, P_4, P_5$, formată din trei segmente de curbă, notate Q_3, Q_4, Q_5. Capătul stâng al segmentului Q_3 se află în vecinătatea punctului de control P_0; capătul dreapta al segmentului Q_5 se află în vecinătatea punctului de control P_5.

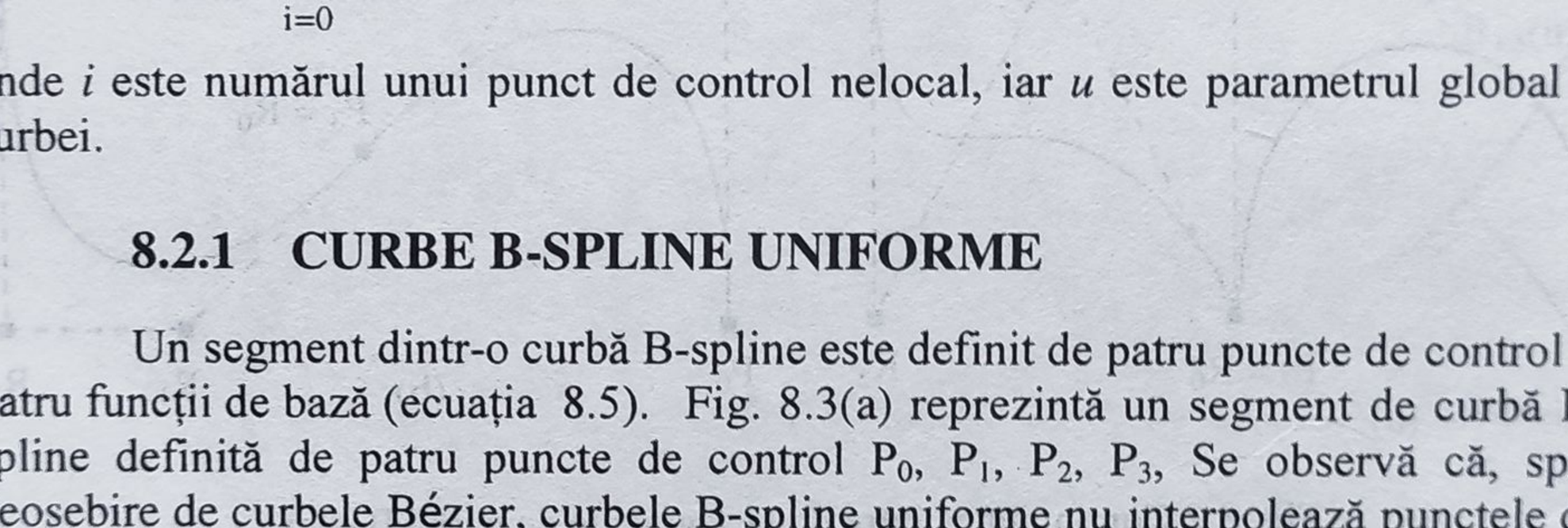


Fig. 8.3 (a) Segment de curbă B-spline.
(b) Curbă B-spline uniformă compusă din trei segmente.

Folosind notația anterioară, se poate descrie organizarea segmentelor curbei din fig. 8.3 (b) astfel:

$\mathbf{Q}_3$ este definit de punctele $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ și funcțiile $\mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$
$\mathbf{Q}_4$ este definit de punctele $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3\mathbf{P}_4$ și funcțiile $\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{B}_4$
$\mathbf{Q}_5$ este definit de punctele $\mathbf{P}_2\mathbf{P}_3\mathbf{P}_4\mathbf{P}_5$ și funcțiile $\mathbf{B}_2\mathbf{B}_3\mathbf{B}_4\mathbf{B}_5$

Faptul că segmentele de curbă adiacente au trei puncte de control comune asigură continuitatea de ordin 0, 1 și 2. Funcțiile de bază B-spline sunt diferite de zero în patru intervale succesive, u_i, u_{i+1}, u_{i+2}, u_{i+3}, centrate la u_{i+2} (fig. 8.4).

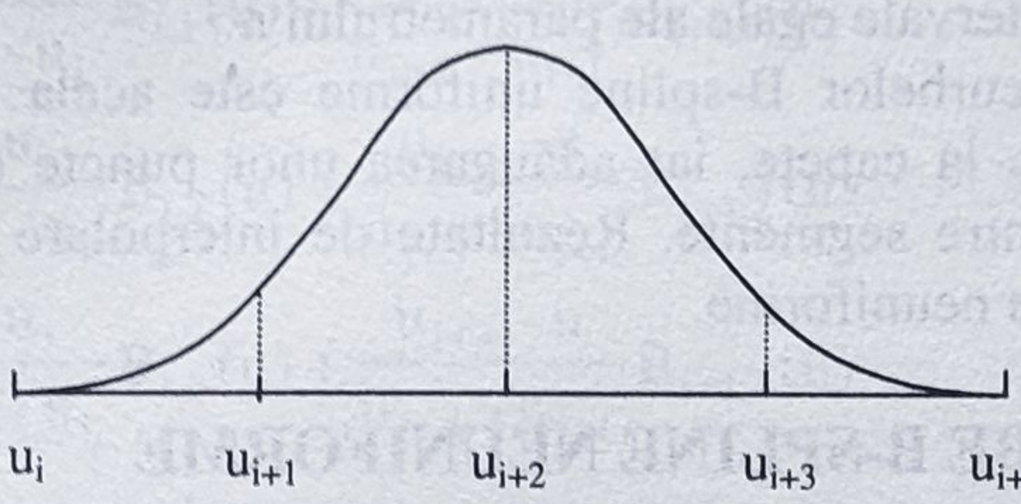


Fig. 8.4 Funcția B-spline $B_i(u)$.

Fiecare funcție de bază B-spline uniformă este o copie translatată a unei astfel de funcții. În intervalul de la u_i la u_{i+1} în care este definit un segment de curbă sunt active patru funcții de bază, iar suma lor este egală cu 1 (fig. 8.5).

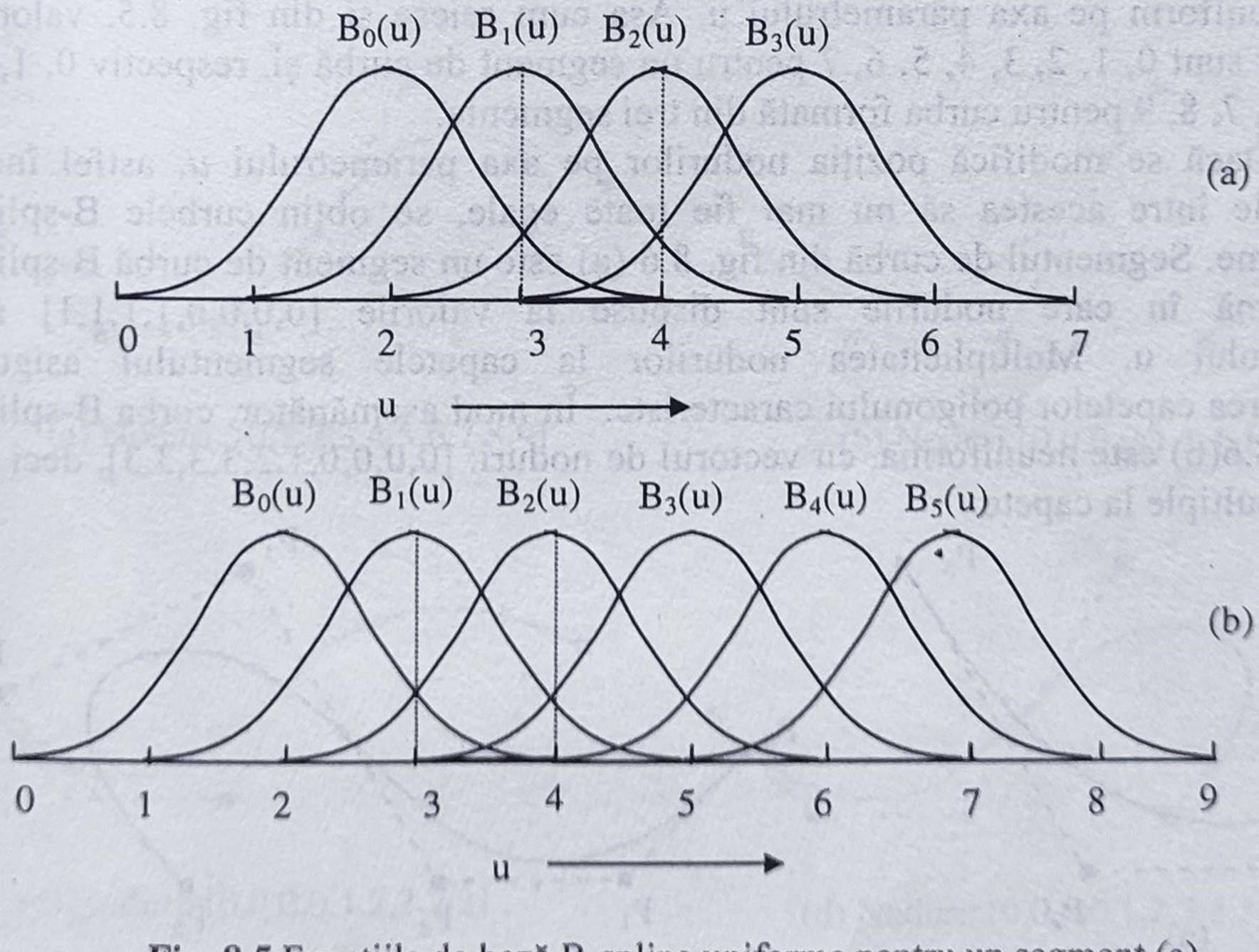


Fig. 8.5 Funcțiile de bază B-spline uniforme pentru un segment (a) și pentru o curbă formată din trei segmente (b).

Punctele de adiacență între funcțiile de bază se numesc *noduri* (*knots*) şi ele se specifică prin valoarea parametrului u la care o funcție de bază devine inactivă şi alta devine activă. Pentru un segment de curbă sunt folosite patru funcții de bază şi fiecare funcție este activă pe patru intervale ale parametrului u, deci un segment este definit peste 8 noduri (fig. 8.5(a)). O curbă B-spline la care intervalele între noduri sunt egale se numeşte curbă B-spline uniformă.

Se poate rezuma că o curbă B-spline este compusă din $m - 2$ segmente, definite de $m + 1$ funcții de bază, peste $m + 5$ noduri. Pentru curba din fig. 8.3(b), $m = 5$, deci sunt 6 puncte de control şi 6 funcții de bază, 3 segmente de curbă şi 10 noduri, distribuite la intervale egale ale parametrului *u*.

Dezavantajul curbelor B-spline uniforme este acela că nu interpolează punctele de control de la capete, iar adăugarea unor puncte de control multiple reduce continuitatea între segmente. Rezultate de interpolare mai bune se obțin folosind curbe B-spline neuniforme.

8.2.2 CURBE B-SPLINE NEUNIFORME

O curbă B-spline neuniformă este o curbă în care intervalele parametrice între noduri succesive nu sunt neapărat egale. Acest lucrù înseamnă că funcțiile de amestec nu mai sunt translatate una față de cealălaltă, ci variază de la un interval la altul. Forma obişnuită a curbelor B-spline neuniforme este aceea în care unele intervale între noduri sunt zero, adică nodurile sunt multiple. Acest lucru permite interpolarea tuturor punctelor de control (capete sau intermediare).

Curbele B-spline din fig. 8.3 sunt curbe uniforme, deoarece nodurile sunt dispuse uniform pe axa parametrului u. Aşa cum reiese şi din fig. 8.5, valorile nodurilor sunt 0, 1, 2, 3, 4, 5, 6, 7 pentru un segment de curbă şi, respectiv 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 pentru curba formată din trei segmente.

Dacă se modifică poziția nodurilor pe axa parametrului *u*, astfel încât intervalele între acestea să nu mai fie toate egale, se obțin curbele B-spline neuniforme. Segmentul de curbă din fig. 8.6 (a) este un segment de curbă B-spline neuniformă în care nodurile sunt dispuse la valorile [0,0,0,0,1,1,1,1] ale parametrului u. Multiplicitatea nodurilor la capetele segmentului asigură interpolarea capetelor poligonului caracteristc. În mod asemănător, curba B-spline din fig. 8.6(b) este neuniformă, cu vectorul de noduri: [0,0,0,0,1,2,3,3,3,3], deci cu noduri multiple la capete.

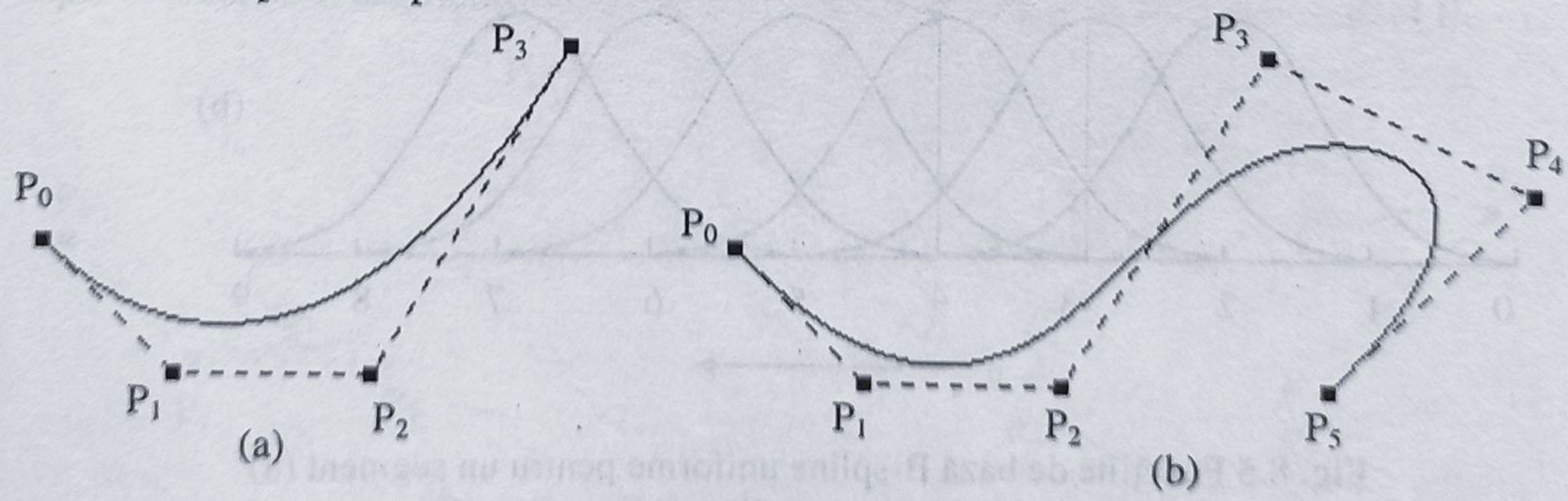


Fig. 8.6 (a) Segment de curbă B-spline neuniformă cu nodurile [0,0,0,0,1,1,1,1].
(b) Curbă B-spline neuniformă cu nodurile [0,0,0,0,1,2,3,3,3,3].

Algoritmul de calcul al funcţiilor de bază B-spline (cunoscut sub numele de algoritmul *Cox-deBoor*) generează recursiv funcţiile de bază B-spline uniforme sau neuniforme de orice grad. Dacă se defineşte $B_{i,j}(u)$ funcţia de bază de ordinul j care ponderează punctul P_i, (ordinul este gradul polinomului plus 1), atunci funcţiile de bază B-spline cubice (de ordin 4) se pot calcula astfel:

$$B_{i,1}(u) = \begin{cases} 1 & \text{pentru } u_i \le u \le u_{i+1} \\ 0 & \text{altfel} \end{cases}$$

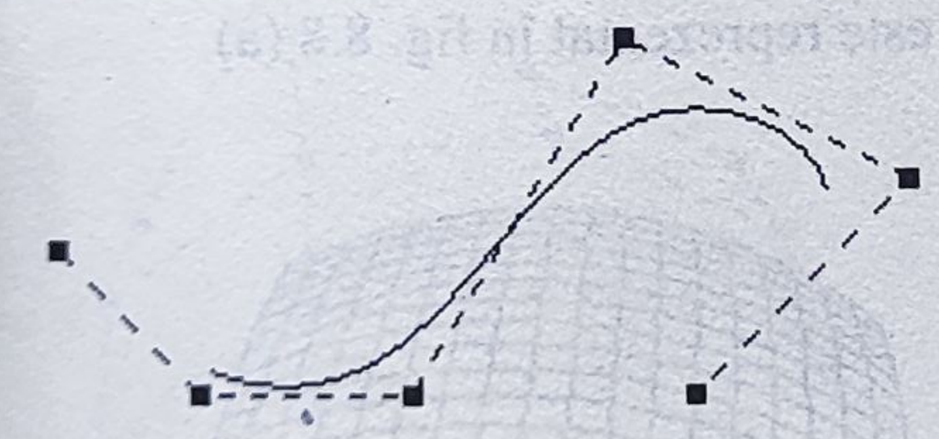
$$B_{i,2} = \frac{u - u_i}{u_{i+1} - u_i} B_{i,1}(u) + \frac{u_{i+2} - u}{u_{i+2} - u_{i+1}} B_{i+1,1}(u)$$

$$B_{i,3} = \frac{u - u_i}{u_{i+2} - u_i} B_{i,2}(u) + \frac{u_{i+3} - u}{u_{i+3} - u_{i+1}} B_{i+1,2}(u)$$

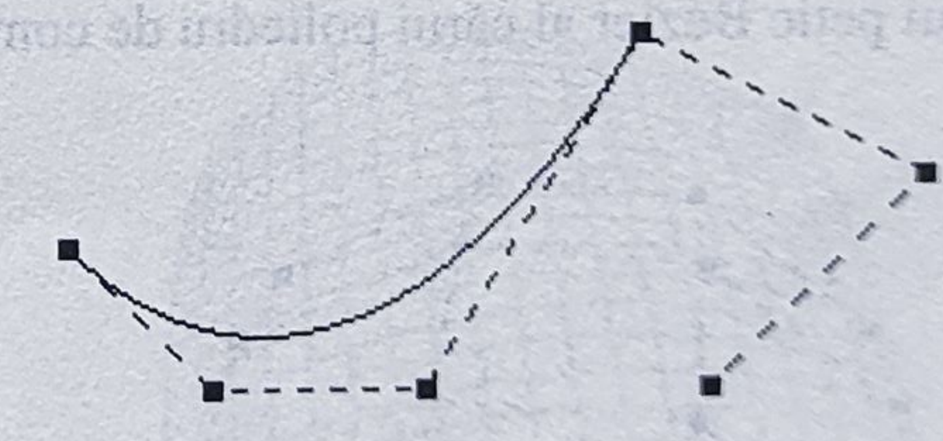
$$B_{i,4} = \frac{u - u_i}{u_{i+3} - u_i} B_{i,3}(u) + \frac{u_{i+4} - u}{u_{i+4} - u_{i+1}} B_{i+1,3}(u) \qquad (8.7)$$

În formulele Cox-deBoor, poate apare împărţirea 0/0, atunci când un nod se repetă; algoritmul atribuie rezultat zero unei astfel de împăţiri prin testarea numărătorului şi, dacă acesta este zero, rezultatul este zero, indiferent de valoarea numitorului.

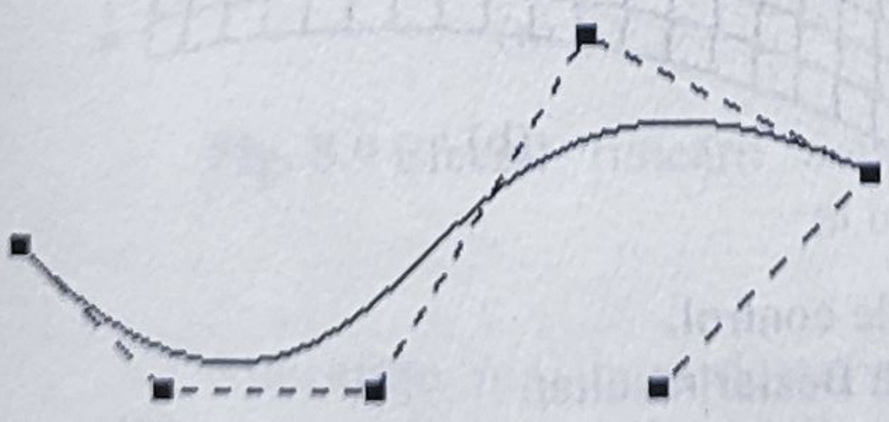
Multiplicarea nodurilor are un efect de asimetrizare a funcţiilor de bază şi acest lucru dă posibilitatea ca un segment de curbă să fie obligat să treacă printr-un punct de control dat. În fig. 8.7 sunt date curbele B-spline care se obţin folosind aceleaşi puncte de control, pentru diferite valori ale nodurilor.



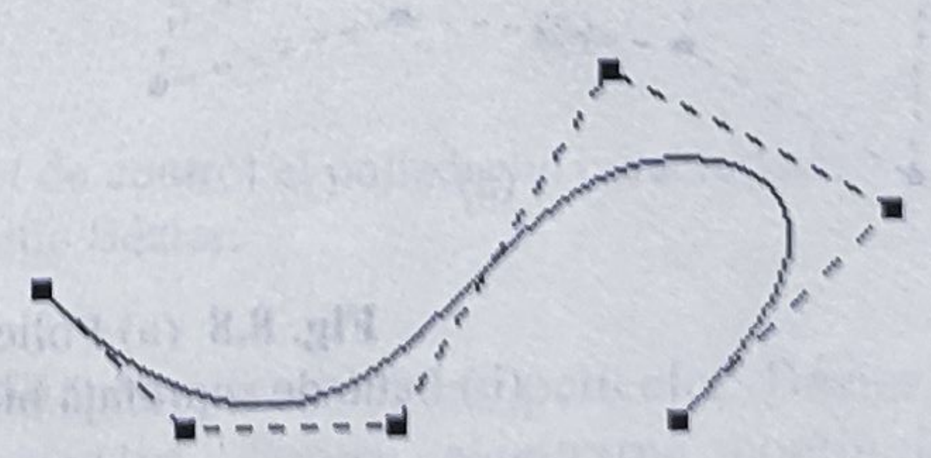
(a) Noduri: [0,1,2,3,4,5,6,7,8,9]



(b) Noduri:[0,0,0,0,1,1,1,1]



(c) Noduri: [0,0,0,0,1,2,2,2,2]



(d) Noduri: [0,0,0,0,1,2,3,3,3,3]

Fig. 8.7 Curbe B-spline pentru diferite valori ale nodurilor.

În primul caz (a), nodurile sunt distribuite uniform şi curba este B-spline uniformă, care nu interpolează punctele terminale. În al doilea caz, s-a obținut un singur segment care interpolează între P_0 şi P_3 şi care este o curbă Bezier. Deci, atât curbele B-spline uniforme, cât şi curbele Bezier sunt cazuri particulare ale curbelor B-spline neuniforme. În cazurile (c) şi (d) nodurile multiple asigură interpolarea între anumite puncte de control. Acestea sunt curbe B-spline neuniforme.

Curbele B-spline neuniforme sunt deosebit de flexibile, permițând reprezentarea unor forme complexe cu un număr relativ mic de funcții de bază de grad fix (cubic în mod obişnuit).

8.3 SUPRAFEȚE BÉZIER

Se poate extinde reprezentarea parametrică cubică a segmentelor de curbă, descrisă în subcapitolele precedente, la reprezentarea peticelor de suprafețe parametrice bicubice. Un petic de suprafață bicubică Bézier este definit prin ecuațiile:

$$Q(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{ij} B_i(u) B_j(v) \qquad (8.8)$$

Un petic Bézier este definit prin 16 puncte de control, care formează o suprafață de control numită poliedru caracteristic. Denumirea care se dă acestei suprafeței este oarecum improprie, dat fiind că este nu este o suprafață închisă, aşa cum este definit un poliedru. În fig. 8.8 (b) este prezentată imaginea wireframe a unui petic Bézier al cărui poliedru de control este reprezentat în fig. 8.8 (a).

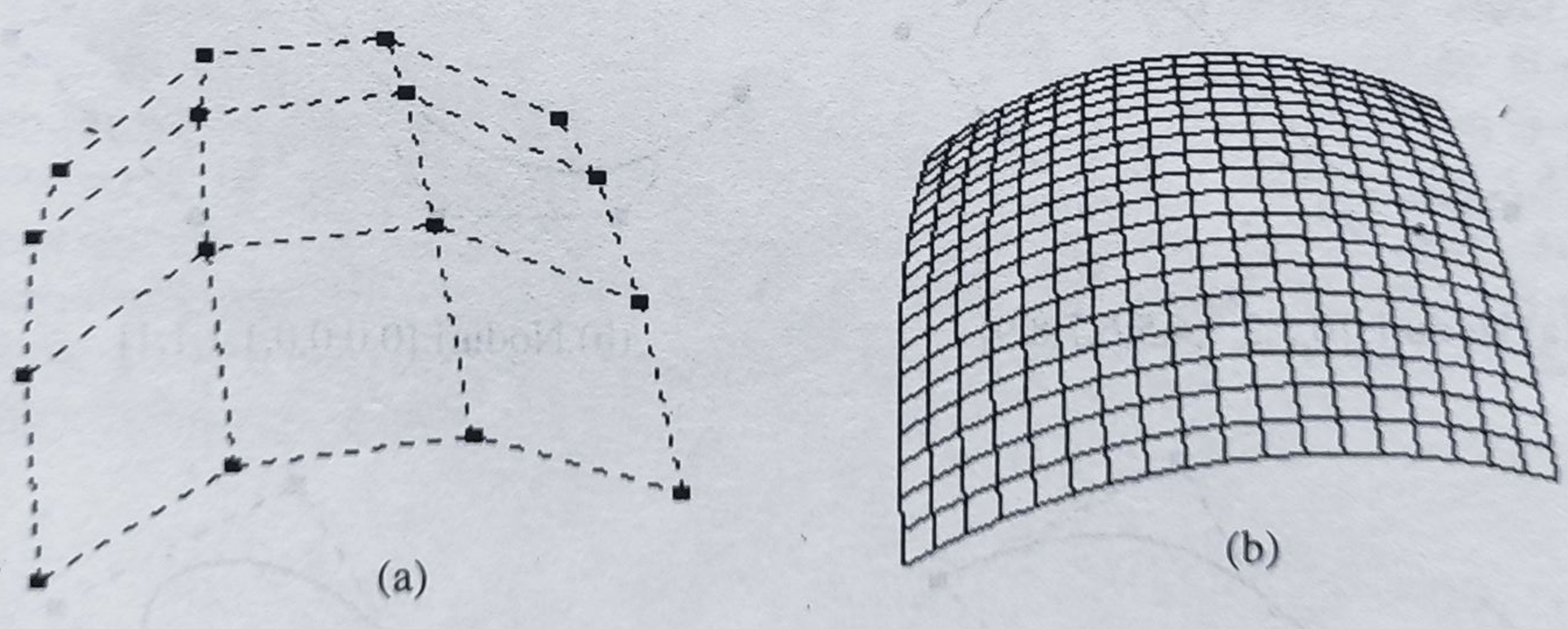


Fig. 8.8 (a) Poliedrul de control.
(b) Petic de suprafață bicubică Bézier rezultat.

Proprietățile curbelor Bézier se extind în domeniul suprafețelor. Figura 8.9 arată cum se deformează un petic atunci când unul din vârfurile poliedrului de control este "tras" în sus.

Modelul unui petic Bézier se reprezintă prin lista punctelor lui de control în sistemul de referință local (de modelare). În forma cea mai simplă, memorat ca un tablou de $4 \times 4 \times 3$ numere reale, modelul peticului din fig. 8.8 este:

```
double points[4][4][3] = {
      {{-3.0,-1.0, 4.0},
      {-1.0, 1.0, 4.0},
      { 1.0, 1.0, 4.0},
      { 3.0,-1.0, 4.0}},
      {{-3.0,-0.0, 2.0},
      {-1.0, 2.0, 2.0},
      { 1.0, 2.0, 2.0},
      { 3.0,-0.0, 2.0}},
      {{-3.0,-0.0, 0.0},
      {-1.0, 2.0, 0.0},
      { 1.0, 2.0, 0.0},
      { 3.0,-0.0, 0.0}},
      {{-3.0,-1.0, -2.0},
      {-1.0, 1.0, -2.0}, //  se modifica la y = 3
      { 1.0, 1.0, -2.0},
      { 3.0,-1.0, -2.0}},
};
```

Modificarea coordonatei y a celui de-al doilea punct din ultimul poligon de control (marcată în comentariu) produce modificarea prezentată în fig. 8.9.

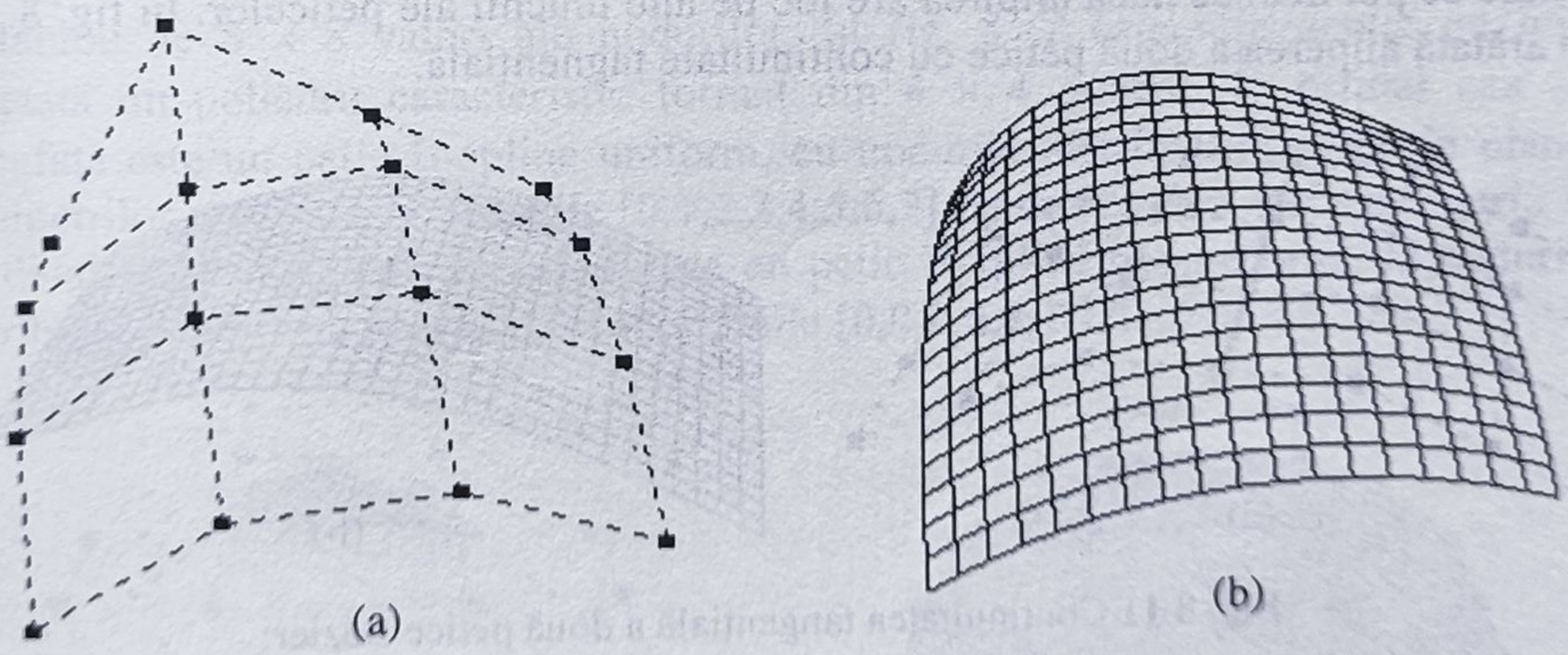


Fig. 8.9 Efectul "ridicării" unui punct de control al poliedrului caracteristic al unui petic Bézier.

Suprafețe mai complexe se obțin prin combinarea peticelor Bézier, cu condiția menținerii continuității între acestea. Pentru asigurarea continuității poziționale (de ordin 0) între două petice $\mathbf{Q}(u, v)$ și $\mathbf{R}(u, v)$ trebuie să fie îndeplinită una din condițiile:

$$\mathbf{Q}(1,v) = \mathbf{R}(0,v) \text{ sau } \mathbf{Q}(0,v) = \mathbf{R}(1,v), \text{ pentru } 0 \le v \le 1 \text{ sau}$$
$$\mathbf{Q}(u,1) = \mathbf{R}(u,0) \text{ sau } \mathbf{Q}(u,0) = \mathbf{R}(u,1), \text{ pentru } 0 \le u \le 1.$$

Această condiţie înseamnă cele două petice au patru puncte de control comune (deci poliedrele caracteristice ale celor două petice au trei muchii comune). În fig. 8.10 este prezentată imaginea a două petice Bézier alipite cu continuitate poziţională. Se poate observa curbura accentuată care apare la această alipire.

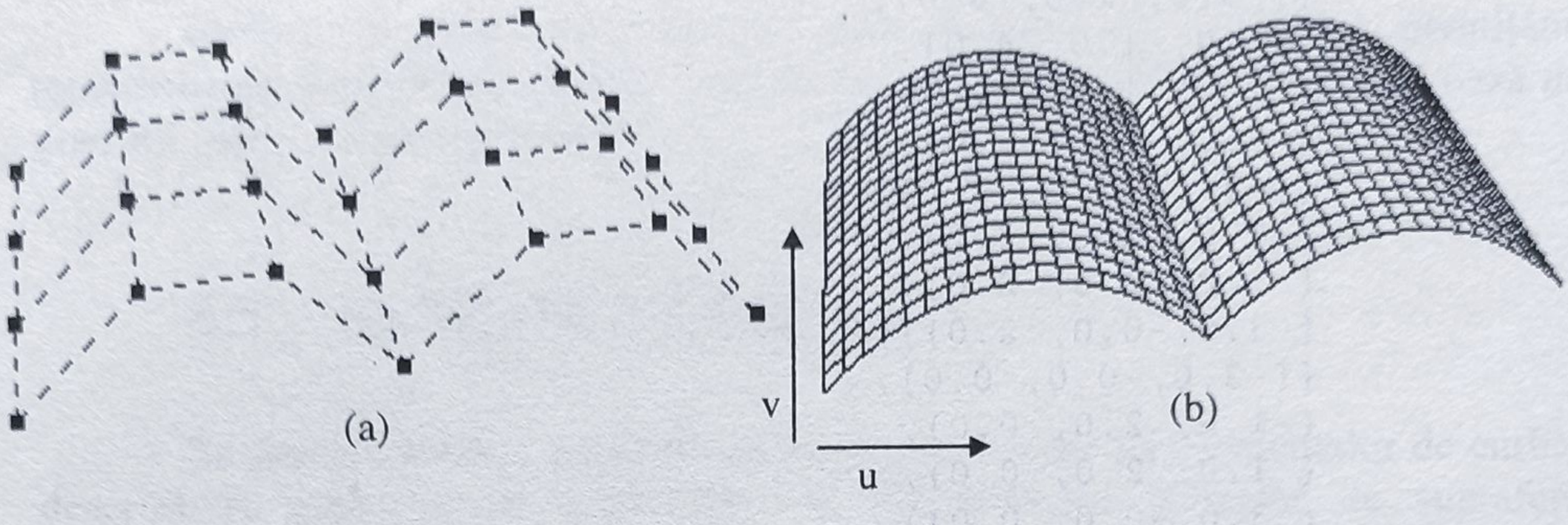


Fig. 8.10 Continuitatea poziţională între două petice Bézier:
(a) poliedrele caracteristice; (b) peticele Bézier.

Pentru satisfacerea *continuităţii tangenţiale* (de ordinul 1) trebuie ca vectorul tangent la primul petic la $u = 1$ să aibă aceeaşi valoare cu vectorul tangent la cel de-al doilea petic la $u = 0$, pentru toate valorile lui v cuprinse în intervalul [0,1], dacă alipirea se face de-a lungul muchiei cu $u = 1$ a primului petic. Condiţii similare se pot deduce dacă alipirea are loc pe alte muchii ale peticelor. În fig. 8.11 este arătată aliperea a două petice cu continuitate tagnenţială.

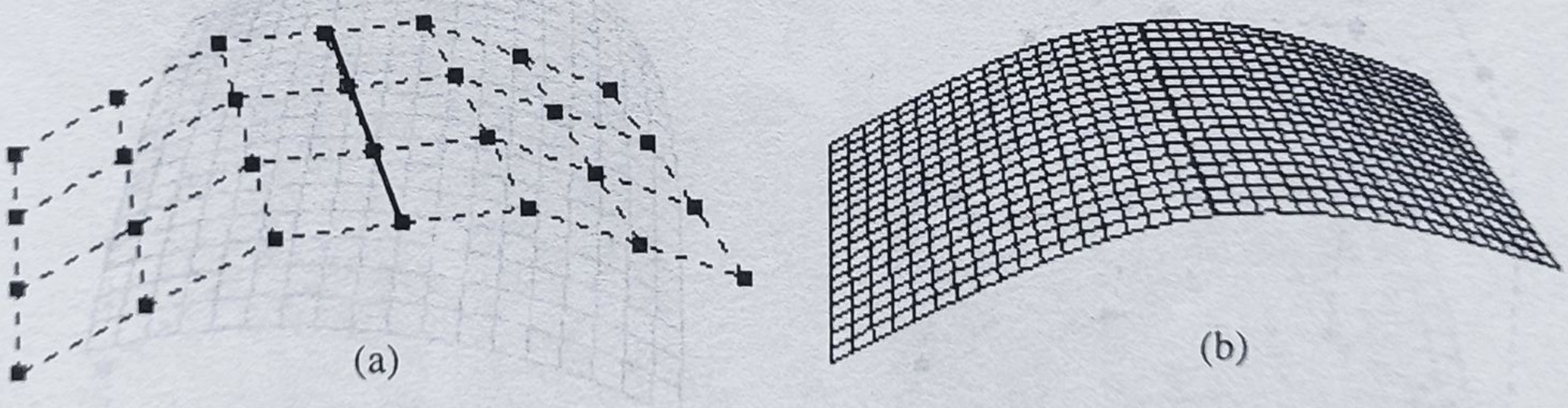


Fig. 8.11 Continuitatea tangenţială a două petice Bézier:
(a) poliedrele caracteristice; (b) peticele Bézier.

Condiţia de continuitate tangenţială este foarte restrictivă, deoarece necesită ca 4 grupe de câte 3 puncte de control să fie coliniare. La adăugarea unui petic nou, 4 puncte de control sunt deja fixate (punctele de alipire), iar alte 4 trebuie să se afle pe direcţii impuse de peticul precedent pentru asigurarea coliniarităţii respective. Deci rămân puţine puncte de control a formei peticului, ceea ce reprezintă o dificultate deosebită de proiectare. O condiţie ceva mai puţin restrictivă, dar acceptabilă de alipire a două petice, dezvoltată de Bézier în 1972, este continuitatea tangentelor pe frontiera de adiacenţă.

Mai trebuie menționat faptul că nu orice formă poate fi aproximată prin rețele de petice patrulatere, precum cele descrise anterior. De exemplu, la modelarea unei sfere (sau elipsoid), la poli peticele patrulatere degenerează în petice triunghiulare.

8.4 SUPRAFEȚE B-SPLINE

Un petic bicubic B-spline este definit de ecuația:

$$\mathbf{Q}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{P}_{ij} \mathbf{B}_{ij}(u, v) \tag{8.9}$$

unde $\mathbf{P}_{ij}$ este un tablou de puncte de control, iar $\mathbf{B}_{ij}(u)$ este o funcție de bază de două varaiabile u şi v, care se poate genera prin produsul funcțiilor de bază ale curbelor B-spline:

$$\mathbf{B}_{i,j}(u,v) = \mathbf{B}_i(u)\mathbf{B}_j(v)$$

Peticele B-spline se consideră rectangulare, definite prin poliedrul caracteristic şi printr-un tablou de noduri care formează o grilă în planul parametrilor u şi v.

Analog definirii curbelor, un petic de suprafață B-spline poate fi compus din mai multe segmente de petice, fiecare segment de petic necesitând o grilă de 4×4 puncte de control care sunt combinate cu 4×4 funcții de bază, definite peste un tablou de 8×8 valori ale nodurilor. În fig. 8.12 este reprezentat cu linie punctată un poliedru caracteristic format din 4×4 puncte. În primul caz (a) suprafața este un petic B-spline uniform, cu nodurile distribuite regulat în planul parametrilor u şi pe v în pozițiile [0,1,2,3,4,5,6,7] pentru fiecare din parametri.. În cel de-al doileal caz (b), suprafața este un petic B-spline neunuiform, cu nodurile distribuite neuniform pe u şi pe v în pozițiile [0,0,0,0,1,1,1,1].

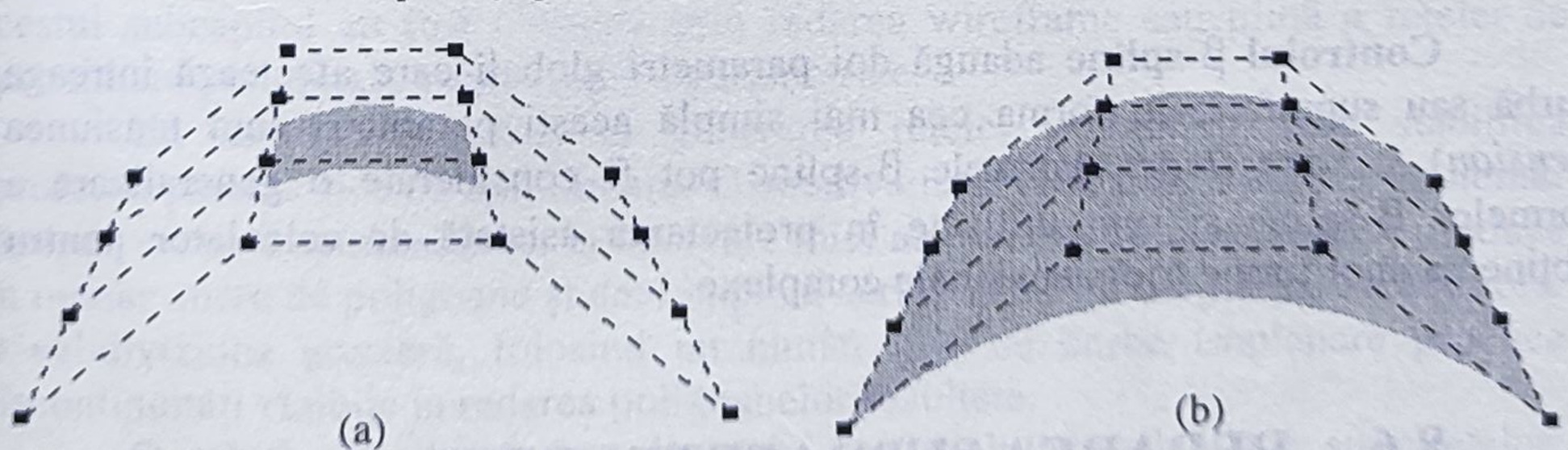


Fig. 8.12 Suprafețe B-spline: (a) petic B-spline uniform; (b) petic B-spline neuniform.

La fel ca şi în cazul curbelor B-spline, o suprafață B-spline uniformă nu interpolează punctele marginale ale poliedrului caracteristic (fig. 8.12 (a)). Multiplicitatea nodurilor în cazul suprafețelor B-spline neuniforme asigură interpolarea punctelor marginale ale poliedrului caracteristic (fig. 8.12(b)).

Suprafeţe complexe se obţin din mai multe segmente de petice alăturate, prin extinderea poliedrului caracteristic şi a tabloului nodurilor.

8.5 EXTINDEREA CONTROLULUI PARAMETRIC: NURBS ŞI β-SPLINE

Două importante extinderi ale curbelor şi suprafeţelor B-spline permit controlul formei atât prin poziţia punctelor de control, cât şi prin alţi parametri.

Curbele şi suprafeţele B-spline neuniforme raţionale (NURBS –*Non-Uniform Rational B-Spline*) extind curbele şi suprafeţele B-spline neuniforme prin adăugarea unui parametru suplimentar fiecărui punct de control. Fiecare punct de control se reprezintă într-un sistem cu 4 coordonate, coordonata suplimentară w fiind un parametru care ponderează efectul fiecărui punct de control.

O curbă NURBS este definită prin puncte de control reprezentate cu patru dimensiuni:

$$P_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$$

Proiecţia perspectivă a unei astfel de curbe în spaţiul tridimensional se numeşte curbă B-spline raţională şi este definită prin ecuaţia:

$$Q(u) = \sum_{i=0}^{n} P_i R_i(u), \text{ unde } R_i(u) = \frac{B_i(u) w_i}{\sum_{j=0}^{n} B_j(u) w_j} \qquad (8.10)$$

Coordonata w este coordonata omogenă şi este folosită pentru diferite transformări geometrice ale punctelor poligonului caracteristic.

Curbele şi suprafeţele B-spline raţionale au aceleaşi proprietăţi ca şi curbele B-spline neuniforme neraţionale dacă w =1.

Controlul β-spline adaugă doi parametri globali care afectează întreaga curbă sau suprafaţă; în forma cea mai simplă aceşti parametri sunt tensiunea (*tension*) şi baza (*bias*). Formele β-spline pot fi considerate o generalizare a formelor B-spline şi sunt utilizate în proiectarea asistată de calculator pentru obţinerea unor forme tridimensionale complexe.

8.6 REDAREA SUPRAFEŢELOR PARAMETRICE

Redarea suprafeţelor parametrice se poate face prin două metode:

- Aproximarea suprafeţei parametrice printr-o reţea de poligoane planare şi redarea acestora folosind procedurile "standard" de redare.
- Redarea directă, din descrierea parametrică a suprafeţelor.

În mod obişnuit, prima modalitate, prin aproximare poligonală, este cea mai eficientă şi este cel mai frecvent folosită.

Se pune întrebarea justificată, de ce să fie folosită reprezentarea parametrică care modelează un obiect cu o precizie ridicată, ca apoi redarea să fie efectuată prin aproximare (mai puţin precisă) cu feţe poligonale. Există două motive pentru care se procedează în acest mod. Primul motiv este acela că în sistemele de proiectare interactive reprezentarea parametrică permite flexibilitate şi simplitate în definirea formelor obiectelor, calităţi care nu se obţin uşor dacă proiectantul este obligat să introducă fiecare poligon separat. Cel de-al doilea motiv este că, de la un model precis reprezentat parametric, se poate trece la aproximarea poligonală cu controlul complet asupra dimensiunii poligoanelor şi deci a preciziei de aproximare. Acest lucru permite, pe de o parte, modelarea adaptivă, prin introducerea unui număr mai mare de poligoane în regiunile cu curbură accentuată a obiectului şi, pe de altă parte, obţinerea unui model cu nivele multiple de detaliu pornind de la o singură reprezentare, reprezentarea parametrică a obiectului.

Redarea directă a suprafeţelor parametrice este dificilă deoarece nu este posibilă aplicarea tehnicilor de calcul eficient pentru conversia unui petic de suprafaţă din descrierea parametrică în mulţimea de pixeli care corespund imaginii acestora. Pentru poligoane planare, operaţia de conversie (care a fost prezentată în capitolul 5) se implementează eficient, prin interpolare atât pentru coordonatele în planul de afişare, cât şi pentru adâncimi, umbrire etc. Pentru suprafeţele parametrice nu se pot aplica astfel de tehnici şi redarea necesită calcule complexe pentru fiecare element al suprafeţei parametrice, ceea ce reprezintă un consum de timp de calcul important. De aceea, redarea directă a suprafeţelor parametrice este rareori folosită în aplicaţiile grafice interactive sau de timp real, fiind abordată în aplicaţii de creare a unor imagini statice foarte precise a obiectelor.

Aproximarea unei suprafeţe parametrice cu o reţea de poligoane planare se obţine folosind curbe izoparametrice (cu valoare constantă a unui parametru). Prin intersecţia curbelor izoparametrice se generează o reţea de puncte ale suprafeţei care se folosesc ca vârfuri ale reţelei de poligoane planare. Imaginile din figurile acestui subcapitol au fost obţinute prin redarea wireframe sau plină a reţelei de aproximare poligonală a suprafeţelor parametrice.

Un aspect important în generarea reţelei poligonale este stabilirea rezoluţiei parametrice de aproximare. O subdiviziune prea fină, folosind un număr mare de curbe isoparametrice, la intervale mici ale parametrului respectiv, produce un număr mare de poligoane şi deci implică cerinţe mari de timp redare a acestora. O subdiviziune grosieră, folosind un număr mic de curbe isoplanare produce discontinuităţi vizibile la redarea poligoanelor rezultate.

O soluţie mai bună este divizarea cu rezoluţie variabilă a suprafeţelor: zonele de suprafaţă cu o curbură accentuată sunt divizate pe intervale mai mici, iar zonele cu curbură locală mică sunt divizate pe intervale mai mari. Problemele care apar în cazul divizării cu rezoluţie variabilă (posibilitatea apariţiei "găurilor") sunt soluţionate prin înlocuirea evaluării punctelor pe curbele izoparametrice cu subdivizarea acestora prin introducerea unor puncte de control suplimentare [Watt95].

8.7 FUNCŢII OPENGL PENTRU REDAREA SUPRAFEŢELOR PARAMETRICE

În OpenGl sunt prevăzute posibilităţi de generare şi redare a curbelor şi suprafeţelor Bézier folosind funcţii numite evaluatori, care calculează coordonatele punctelor de pe o curbă sau o suprafaţă Bézier, în funcţie de valorile parametrilor (u sau u şi v). Aceste puncte sunt grupate în primitive geometrice, care sunt redate pe display. Funcţiile de evaluare a coordonatelor sunt `glEvalCoord1#()` şi `glEvalCoord2#()` pentru curbe, respectiv suprafeţe Bézier.

8.7.1 GENERAREA ŞI REDAREA CURBELOR BÉZIER

Programul OpenGL prin care s-a obţinut prima curbă Bézier din fig. 8.1 este dat în exemplul următor.

■ **Exemplul 8.1**

```
#include <GL/glut.h>
#define M 20
GLfloat points[4][3] = {
      { -1.0, -1.0, 0.0},
      { -1.0,  1.0, 0.0},
      {  1.0,  1.0, 0.0},
      {  1.0, -1.0, 0.0}
};
void Init(void){
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glPointSize(5);
   glLineStipple(1, 0x0F0F);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_FLAT);
}
void Display(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glEnable(GL_MAP1_VERTEX_3);
   glColor3f(0.0, 0.0, 0.0);
   glPushMatrix();
   glTranslated(0,0,-7);
   glDisable(GL_LINE_STIPPLE);
   glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,4,
           &points[0][0]);
   glBegin(GL_LINE_STRIP);
   for (int i = 0; i <= M; i++)
      glEvalCoord1f((GLfloat)i/M);
   glEnd();
   // Afisare puncte de control
   glPointSize(5);
   glBegin(GL_POINTS);
```

```
    for (i = 0; i < 4; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
     // Afisare poligon caracteristic
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 4; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,(GLfloat)w/(GLfloat)h,0.1,1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Bezier");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

Programul afişează un segment de curbă Bézier cu linie plină, punctele de control ca mici pătrate şi poligonul caracteristic cu linie punctată (prin validarea `glEnable(GL_LINE_STIPPLE)` şi setarea succesiunii de linii întrerupte cu funcţia `glLineStipple()`).

Cele patru punctele de control ale unei curbe Bézier sunt memorate în tabloul `points[4][3]` care este unul din argumentele funcţiei `glMap1f()` care defineşte tipul de corespondenţă între datele de intrare şi datele calculate de funcţia de evaluare `glEvalCoord1#()`. Prototipurile funcţiei `glMap1#()` sunt următoarele:

```
void glMap1d(GLenum target,GLdouble u1,GLdouble u2,
     GLint stride,GLint order,const GLdouble *points);
void glMap1f(GLenum target,GLdouble u1,GLdouble u2,
     GLint stride,GLint order,const GLfloat *points);
```

Argumentul `target` specifică tipul de puncte de control prevăzute în vectorul `points` şi ce tip de date de ieşire va genera funcţia de evaluare `glEvalCoords()` care va fi apelată ulterior. Acest argument poate lua mai multe valori de constante simbolice, specificând coordonate de vârfuri (`GL_MAP1_VERTEX3`), normale (`GL_MAP1_NORMAL`) sau coordonate de textură (`GL_MAP_TEXTURE_COORD_1,.....,GL_MAP_TEXTURE_COORD_4`).
Intervalul u1 – u2 este intervalul care se mapează liniar în intervalul [0,1] al parametrului u pentru calculul expresiei polinomiale **Q**(u') (relaţia 8.3), unde u' = (u – u1)/(u2 – u1):

$$\mathbf{Q}(u') = \sum_{i=0}^{n} \mathbf{P}_i \mathbf{B}_{i,n}(u')$$

$B_{i,n}(u)$ sunt polinoamele lui Bernstein de gradul n, iar P_i sunt punctele de control memorate în vectorul `points`.

Argumentul `stride` specifică numărul de valori în virgulă flotantă cu care se avansează în vectorul `points` de la un punct de control la următorul (în exemplul de mai sus are valoarea 3). Argumentul `order` reprezintă ordinul curbei, şi este egal cu gradul polinomului Bernstein plus 1. Vectorul `points` conţine punctele de control care sunt folosite de funcţia de evaluare. În felul acesta sunt definite toate datele necesare evaluării relaţiei (8.3): gradul polinomului Bernstein, punctele de control şi intervalul de mapare al parametrului `u`.

Pentru validarea unui anumit tip de evaluare se apelează mai întâi funcţia `glEnable()` cu argument corespunzător argumentului `target` al funcţiei `glMap1#()` (deci poate fi `GL_MAP1_VERTEX3`, etc.).

Funcţia `glEvalCoord1f(GLfloat u)` calculează coordonatele unui punct pe curbă pentru valorea dată a parametrului `u` şi lansează comenzi OpenGL corespunzătoare tipului punctelor de control. De exemplu, dacă punctele de control sunt coordonate de vârfuri (`GL_MAP1_VERTEX3`), atunci, pentru fiecare valoare a parametrului u, coordonatele calculate sunt folosite ca argumente ale unei funcţii `glVertex#()`.

Forma şi dimensiunea segmentului de curbă Bézier se poate modifica prin modificarea poziţiei punctelor de control. Familia de curbe din fig. 8.1(a) s-a obţinut prin modificarea poziţiei punctului de control P_1. La fel, imaginile din fig. 8.13 se obţin dacă se rulează acelaşi program, dar cu vectorul de puncte modificat astfel:

```
GLfloat points[4][3] = {
        { -1.0, -1.0, 0.0},
        {  0.0,  1.0, 0.0},
        {  1.0, -1.0, 0.0},
        {  2.0,  1.0, 0.0},};
```

Curba este aproximată printr-un număr de M segmente de dreaptă, M fiind stabilit în programul de aplicaţie în funcţie de precizia de redare dorită. Dacă numărul de segmente de aproximare este redus, se observă eroarea de aproximare a curbei (fig. 8.13 (b)).

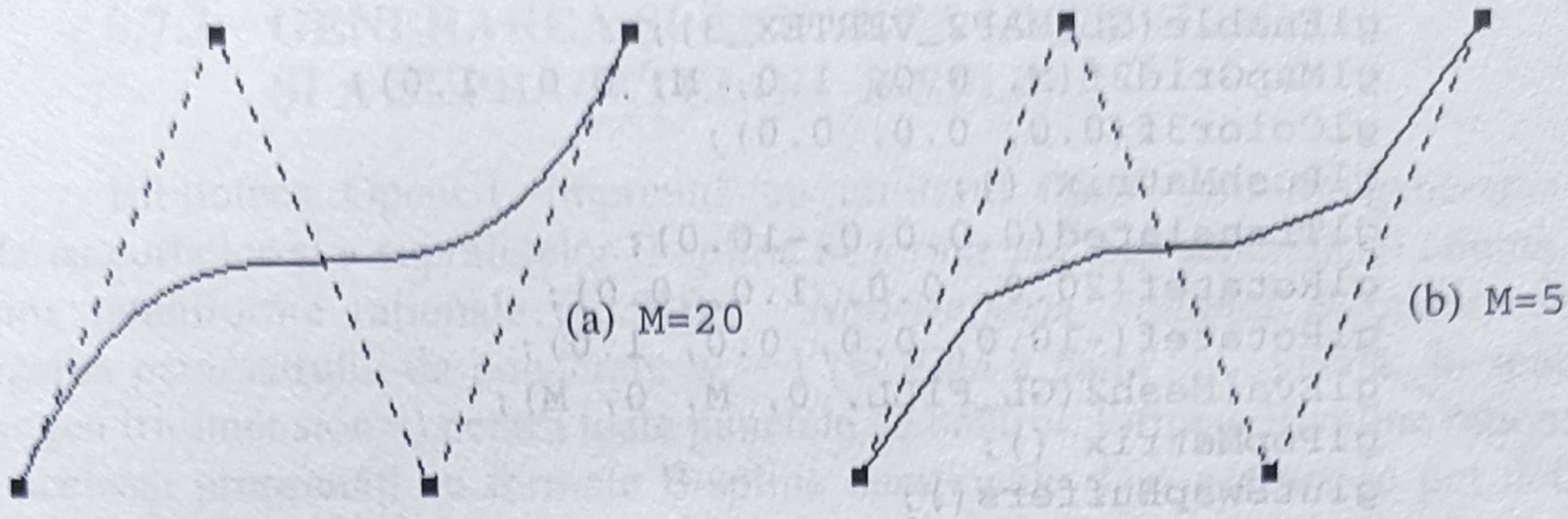


Fig. 8.13 Curbă Bézier cubică, punctele de control şi poligonul caracteristic.

8.7.2 GENERAREA ŞI REDAREA SUPRAFEŢELOR BÉZIER

Pentru redarea suprafeţelor Bézier se apelează o funcţie de evaluare bidimensională `glEvalCoord2#()` care calculează un punct de pe suprafaţă în funcţie de doi parametri u şi v. În exemplul următor se poate urmări modul de redare a unui petic de suprafaţă bicubică Bézier.

■ **Exemplul 8.2**

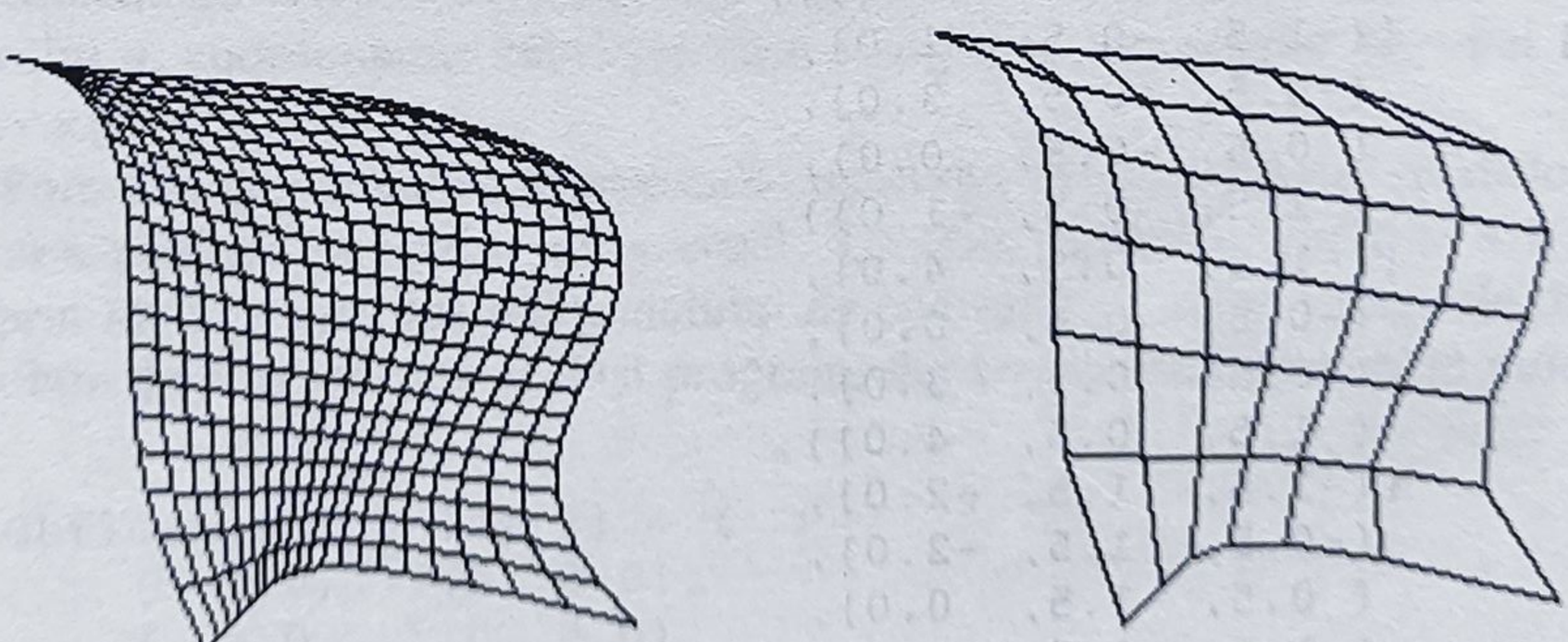
```
#include <GL/glut.h>
#define M 20
GLfloat points[4][4][3] = {
    {{-1.5, -1.5,  4.0},
     {-0.5, -1.5,  2.0},
     { 0.5, -1.5, -1.0},
     { 1.5, -1.5,  2.0}},
    {{-1.5, -0.5,  1.0},
     {-0.5, -0.5,  3.0},
     { 0.5, -0.5,  0.0},
     { 1.5, -0.5, -1.0}},
    {{-1.5,  0.5,  4.0},
     {-0.5,  0.5,  0.0},
     { 0.5,  0.5,  3.0},
     { 1.5,  0.5,  4.0}},
    {{-1.5,  1.5, -2.0},
     {-0.5,  1.5, -2.0},
     { 0.5,  1.5,  0.0},
     { 1.5,  1.5, -1.0}}
};
void Display(){
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
            0, 1, 12, 4, &points[0][0][0]);
```

```
    glEnable(GL_MAP2_VERTEX_3);
    glMapGrid2f(M, 0.0, 1.0, M, 0.0, 1.0);
    glColor3f(0.0, 0.0, 0.0);
    glPushMatrix ();
    glTranslated(0.0,0.0,-10.0);
    glRotatef(20.0, 0.0, 1.0, 0.0);
    glRotatef(-10.0, 0.0, 0.0, 1.0);
    glEvalMesh2(GL_FILL, 0, M, 0, M);
    glPopMatrix ();
    glutSwapBuffers();
}
```

Toate celelalte funcţii ale programului sunt identice cu cele din programul precedent şi nu au mai fost prezentate. La execuţia programului cu parametrul `M=20` se obţine reprezentarea wireframe a suprafeţei Bézier dată prin cele 16 puncte de control memorate în vectorul `points` (fig. 8.14(a)).

Funcţia `glEvalCoord2f(GLfloat u, GLfloat v)` evaluează un punct pe suprafaţă pentru valorile u şi v ale parametrilor curbei şi foloseşte valorile calculate ca argumente ale funcţiilor apelate conform tipului de corespondenţă definit de funcţia `glMap2#()`. Funcţia `glMap2#()` stabileşte modul de evaluare a punctelor de pe o suprafaţă Bézier pentru cei doi parametri. Argumentele de apel sunt similare celor folosite pentru funcţia `glMap1#()`, cu deosebirea că se defineşte ordinul curbei şi intervalul de calcul pentru fiecare parametru.

Suprfaţa curbă este aproximată cu o reţea de poligoane planare şi redată de funcţia `glEvalMesh2()` care specifică modul de redare al poligoanelor (care poate fi `GL_FILL, GL_POINT, GL_LINE`) şi intervalele de eşantionare a suprafeţei pentru parametrii u şi v. Dimensiunile acestor intervale sunt definite prin funcţia `glMapGrid2f()`.



(a) M=20 (b) M=6

Fig. 8.14 Suprafaţă Bézier reprezentată wireframe cu diferite precizii.

În Planşa 4 este redată aceeaşi suprafaţă Bézier cu iluminare şi umbrire.

8.7.3 GENERAREA ŞI REDAREA CURBELOR ŞI A SUPRAFEȚELOR B-SPLINE

Biblioteca OpenGL împreună cu utilitarul GLU, permite generarea şi redarea curbelor şi a suprafeţelor B-spline în forma cea mai generală, şi anume B-spline neuniforme raționale (NURBS - *Non-Uniform Rational B-Spline*). Prin alegerea parametrului de ponderare w = 1 (w fiind a patra coordonată în spațiul omogen tridimensional) pentru toate punctele de control, formele B-spline raționale au aceleaşi proprietăți cu formele B-spline neraționale. Din acestea se pot defini forme B-spline uniforme sau neuniforme în funcție de intervalul parametric al nodurilor.

Funcțiile de evaluare (evaluatori) `glEvalCoord#()` sunt primitive OpenGL care permit evaluarea punctelor de pe curbe şi suprafețe parametrice depinzând de modul de definire a evaluării. Evaluatorii pot fi implementați eficient în hardware. Pentru reprezentarea curbelor şi a suprafețelor B-spline neuniforme raționale sunt necesare funcții mai complexe care utilizează evaluatori OpenGL. Aceste funcții sunt prevăzute de biblioteca auxiliată GLU.

O curbă sau o suprafață NURBS se defineşte ca un obiect de tipul `GLUnurbsObj` care se alocă în memoria liberă (heap). Pentru un astfel de obiect se specifică parametrii de definiție: punctele de control şi vectorul de noduri (pentru unul sau doi parametri). Pentru redarea imaginii unei curbe sau suprafețe NURBS se mai specifică datele de eşantionare, care definesc intervalele la care se evaluează punctele obiectului şi aceste puncte sunt grupate în primitive geometrice (linii sau poligoane). În exemplele următoare este descris pe scurt modul de generare şi redare a curbelor şi suprafețelor NURBS.

■ **Exemplul 8.3**

Programul care urmează generează şi redă o curbă NURBS folosind funcțiile bibliotecii GLU.

```
#include <GL/glut.h>
GLUnurbsObj *theNurb;
GLfloat points[9][3] = {
    { -1.0,  4.0, 0.0},
    { -2.5,  3.0, 0.0},
    { -3.0, -0.5, 0.0},
    { -1.5, -4.0, 0.0},
    {  0.0, -4.0, 0.0},
    {  1.5, -3.0, 0.0},
    {  2.5, -1.5, 0.0},
    {  2.0,  1.0, 0.0},
    {  3.5,  2.0, 0.0}};
void Init(){
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
```

```
        glLineStipple(1,0x0F0F);
        glPointSize(5);
        theNurb = gluNewNurbsRenderer();
    }
    void Display(){
        GLfloat knots[13] = {0,0,0,0,1,2,3,4,5,6,6,6,6};
        gluNurbsProperty(theNurb,
                         GLU_SAMPLING_TOLERANCE, 1.0);
        int nknots = sizeof(knots)/sizeof(GLfloat);
        int npoints = sizeof(points)/sizeof(GLfloat);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(0.0,0.0,0.0);
        glPushMatrix();
        glTranslatef(0.0,0.0,-20.0);
        glDisable(GL_LINE_STIPPLE);
        gluBeginCurve(theNurb);
        gluNurbsCurve(theNurb,      // pointer obiect NURBS
          nknots, knots,            // noduri
          3,                        // interval
          &points[0][0],            // vector puncte control
          4,                        // ordinul curbei
          GL_MAP1_VERTEX_3);        // tip evluator
    gluEndCurve(theNurb);
    // Afisare puncte de control
    glBegin(GL_POINTS);
    for (int i = 0; i < 9; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    // Afisare poligon caracteristic
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 9; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
```

Funcţiile `main()` şi `Reshape()` ale programului sunt identice cu cele din programele precedente. Pentru valorile punctelor de control `points` şi ale nodurilor `knots` de mai înainte se obţine curba din fig. 8.15(a) compusă din şase segmente NURBS, corespunzătoare unui număr de 9 puncte de control şi 13 noduri, dintre care nodurile de la capete sunt noduri cu multiplicitatea 4, pentru interpolarea punctelor de la capete.

Obiectul de tip `GLUnursObj` este creat dinamic în memoria liberă prin apelul funcţiei `gluNewNurbsRenderer()` care returnează pointerul la obiectul creat. Acestui obiect i se atribuie mai multe proprietăţi de redare, dintre care s-a specificat eroarea de eşantionare a curbei la redare prin funcţia `gluNurbsProperty()`.

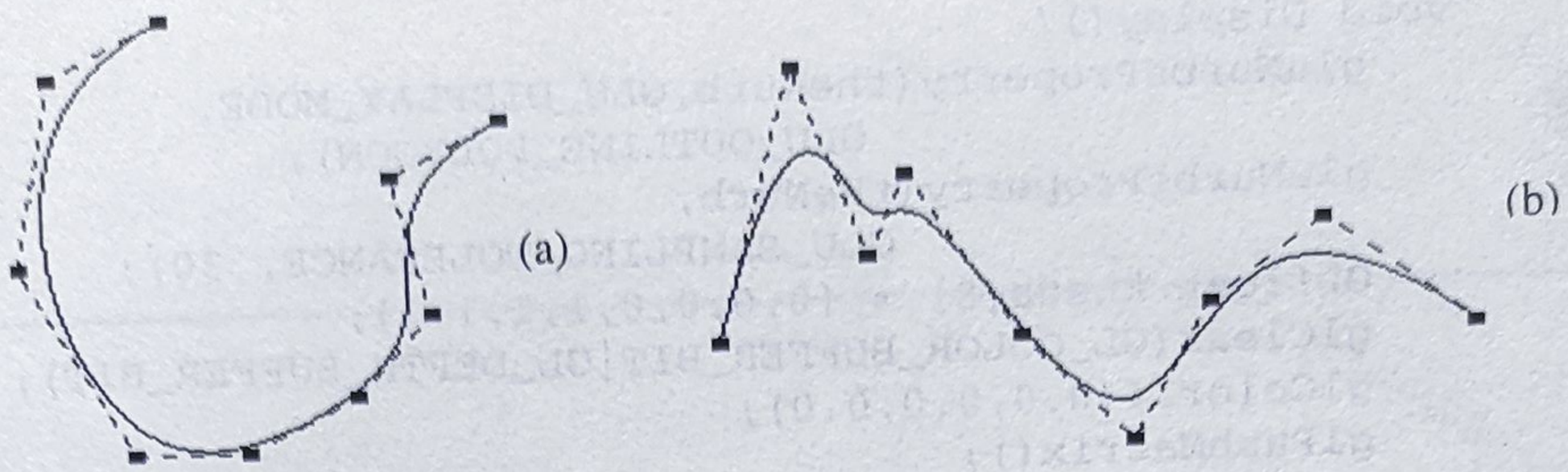


Fig. 8.15 Curbe NURBS

Vectorul punctelor de control (`points`), vectorul nodurilor (`knots`) şi tipul funcţiei de evaluare (`GL_MAP1_VERTEX_3`) sunt parametri care se transmit prin funcţia `gluNurbsCurve()` care generează sau redă o curbă NURBS. Fiecare curbă se redă sau se generează într-un bloc de program cuprins între instrucţiunile `gluBeginCurve()` şi `gluEndCurve()`. Afişarea punctelor de control şi a poligonului caracteristic se face la fel ca în programul precedent. Dacă se execută acelaşi program, cu acelaşi vector de noduri, dar cu un alt vector de puncte de control, se obţine curba din fig. 8.15(b).

■ Exemplul 8.4

Pentru generarea unei suprafeţe NURBS se creează tot un obiect de tipul `GLUmurbsObj`, dar se defineşte un poliedru caracteristic (printr-un tablou de puncte de control) şi doi vectori (care pot fi identici) de noduri, pentru cei doi parametri u şi v ai suprafeţei. Programul de generare a unei suprafeţe NURBS este următorul:

```
#include <GL/glut.h>
GLfloat ctlpoints[4][4][3];
GLUnurbsObj *theNurb;
void init_points(void){
   int u, v;
   for (u = 0; u < 4; u++) {
      for (v = 0; v < 4; v++) {
         ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
         ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);
         if ( (u == 1 || u == 2) && (v == 1 || v == 2))
            ctlpoints[u][v][2] = 3.0;
         else ctlpoints[u][v][2] = -3.0;
      }
   }
}
void Init(){
   glClearColor (1.0, 1.0, 1.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   init_points();
   theNurb = gluNewNurbsRenderer();
}
```

```
void Display(){
   gluNurbsProperty(theNurb,GLU_DISPLAY_MODE,
                            GLU_OUTLINE_POLYGON);
   gluNurbsProperty(theNurb,
                           GLU_SAMPLING_TOLERANCE, 30);
   GLfloat knots[8] = {0,0,0,0,1,1,1,1};
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glColor3f(0.0,0.0,0.0);
   glPushMatrix();
   glTranslatef(0.0,0.0,-20.0);
   glRotatef(330.0, 1.,0.,0.);
   gluBeginSurface(theNurb);
   gluNurbsSurface(theNurb,    // pointer obiect NURBS
      8, knots, 8, knots,      // vector noduri pe u, v
      4*3, 3,                  // intervale pe u, v
      &ctlpoints[0][0][0],     // tablou puncte control
      4, 4,                    // ordin u, v
            GL_MAP2_VERTEX_3);     // tip evaluator
      gluEndSurface(theNurb);
      glPopMatrix();
      glutSwapBuffers();
}
```

Imaginea creată la execuţia acestui program este dată în fig. 8.16(a). Generarea şi redarea suprafeţei este realizată de funcţia `gluNurbsSurface()`, căreia i se transmit ca argumente tabloul de puncte de control (`ctlpoints`), vectorii de noduri (`knots`) şi tipul evaluatorului (`GL_MAP2_VERTEX_3`). Tabloul de puncte de control este generat de funcţia `init_points()`.

Modul de redare a feţelor poligonale prin care se aproximează peticul de suprafaţă este stabilit prin funcţia `gluNurbsProperty()` cu tipul de redare `GLU_OUTLINE_POLYGON` pentru redare wireframe (fig. 8.16 (a)), respectiv `GLU_FILL` pentru redarea plină a feţelor (fig. 8.16 (b)).

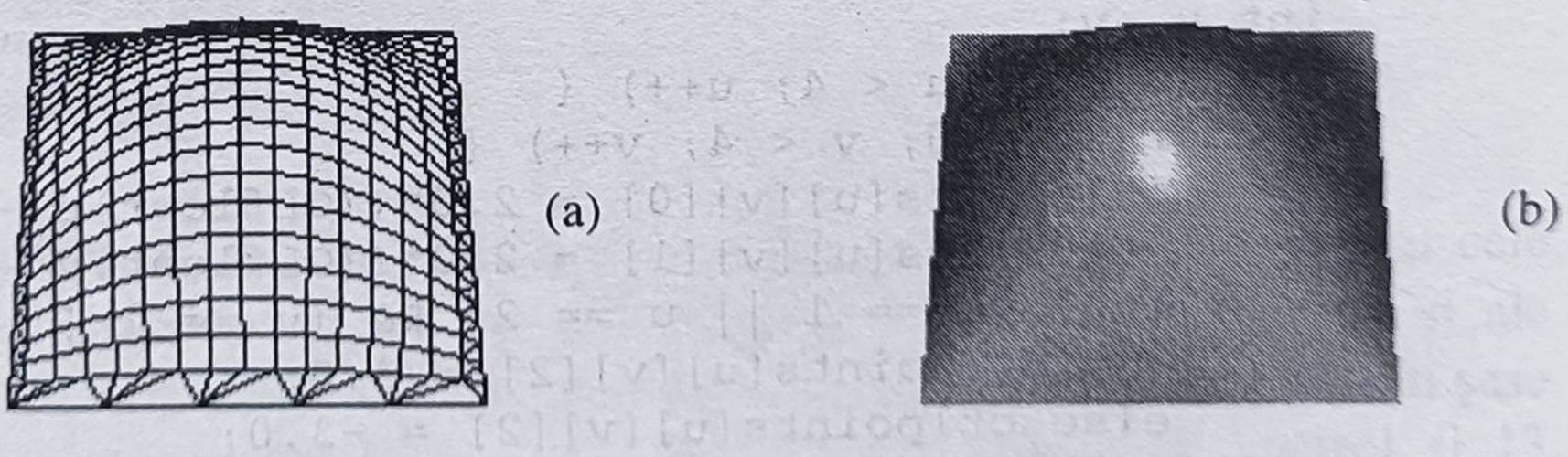


Fig. 8.16 Suprafaţă NURBS redată wireframe (a) şi cu feţe iluminate (b).

În cele două exemple 8.3 şi 8.4 s-au definit forme NURBS cu coordonata de ponderare w = 1, deoarece în evaluarea `GL_MAP1_VERTEX3` sau `GL_MAP2_VERTEX3` punctele de control sunt date cu trei coordonate (x, y, z) şi se consideră coordonata w = 1. Astfel de forme NURBS au aceleaşi proprietăţi ca şi formele B-spline neraţionale neuniforme.

9

ANTI-ALIASING

Se poate observa uşor că liniile afişate pe display apar ca o secvență de segmente care dau un aspect zimțat (în trepte de scară), aspect cunoscut sub numele de *aliasing*. În text se va păstra termenul original de *aliasing*, dat fiind că traducerile sunt mai puțin sugestive. Cauza fundamentală a aliasing-ului în grafica pe calculator este crearea imaginilor prin eşantionare în domeniul spațiului şi, pentru imaginile animate, şi în domeniul timp. Pentru generarea unei imagini, pasul final îl constitue calculul intensității pixelilor, care reprezintă trecerea de la intensitatea $i(x,y)$ definită în spațiul bidimensional continuu, la intensitatea fiecărui pixel în spațiul imagine discret. Această trecere este echivalentă cu eşantionarea spațiului bidimensional continuu cu o grilă de puncte de eşantionare considerate, de exemplu, în centrul fiecărui pixel.

Cea mai familiară manifestare a aliasing-ului o constitue aspectul de rugozitate (*jagged*) al liniilor sau al muchiilor poligoanelor. De asemenea, suprafețele înguste pot apare întrerupte, în funcție de orientarea lor. Un alt efect al aliasing-ului îl constitue apariția şi dispariția obiectelor mici care ocupă o suprafață mai mică decât suprafața unui pixel şi care sunt redate sau nu în funcție de intersecția cu grila de eşantionare. Aceste efecte sunt mai supărătoare în secvențele de imagini în mişcare. Muchiile în formă de scară sau suprafețele întrerupte dau o senzație de mişcare necontrolată a acestora (*crowl*), care este intolerabilă în multe aplicații de realitate virtuală. Tehnicile de prelucrare a imaginii pentru atenuarea efectelor de aliasing sunt cunoscute sub numele de *anti-aliasing*. Aceste tehnici pot fi înțelese şi implementate algoritmic pe baza teoriei Fourier.

9.1 CONSIDERAȚII TEORETICE ASUPRA ALIASING-ULUI

Imaginea bidimensională în sistemul de referință al porții de afişare este considerată un câmp determinist de dimensiuni infinite $i(x, y)$, ale cărei eşantioane se obțin prin multiplicarea cu o funcție de eşantionare spațială $s(x, y)$:

$$s(x,y)=\sum_{n=-\infty}^{+\infty}\sum_{m=-\infty}^{+\infty}\delta(x-n\Delta x,y-m\Delta y) \tag{9.1}$$

Funcţia $s(x, y)$ este compusă dintr-o matrice infinită de funcţii delta, aranjate sub forma unei grile (matrice) de distanţe $(\Delta x, \Delta y)$ (fig. 9.1).

Imaginea eşantionată $f(x,y)$ este:

$$f(x,y)=i(x,y)\,s(x,y)=\sum_{n=-\infty}^{+\infty}\sum_{m=-\infty}^{+\infty}i(n\Delta x,m\Delta y)\,\delta(x-n\Delta x,y-m\Delta y) \tag{9.2}$$

unde funcţia delta bidimensională este $\delta(n,m)=\begin{cases}1 \text{ pentru } n=0,m=0\\0 \text{ pentru } n\neq 0,m\neq 0\end{cases}$

Reprezentarea în domeniul frecvenţei spaţiale ω (radiani/distanţă) a imaginii eşantionate se obţine prin aplicarea *transformatei Fourier bidimensionale* continue imaginii eşantionate:

$$F(\omega_x,\omega_y)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}f(x,y)e^{-j(\omega_x x+\omega_y y)}\,dx\,dy \tag{9.3}$$

Conform teoremei convoluţiei, transformata Fourier a imaginii eşantionate poate fi exprimată prin produsul de convoluţie dintre spectrul imaginii $I(\omega_x, \omega_y)$ şi spectrul funcţiei de eşantionare $S(\omega_x, \omega_y)$:

$$F(\omega_x,\omega_y)=\frac{1}{4\pi^2}I(\omega_x,\omega_y)\otimes S(\omega_x,\omega_y) \tag{9.4}$$

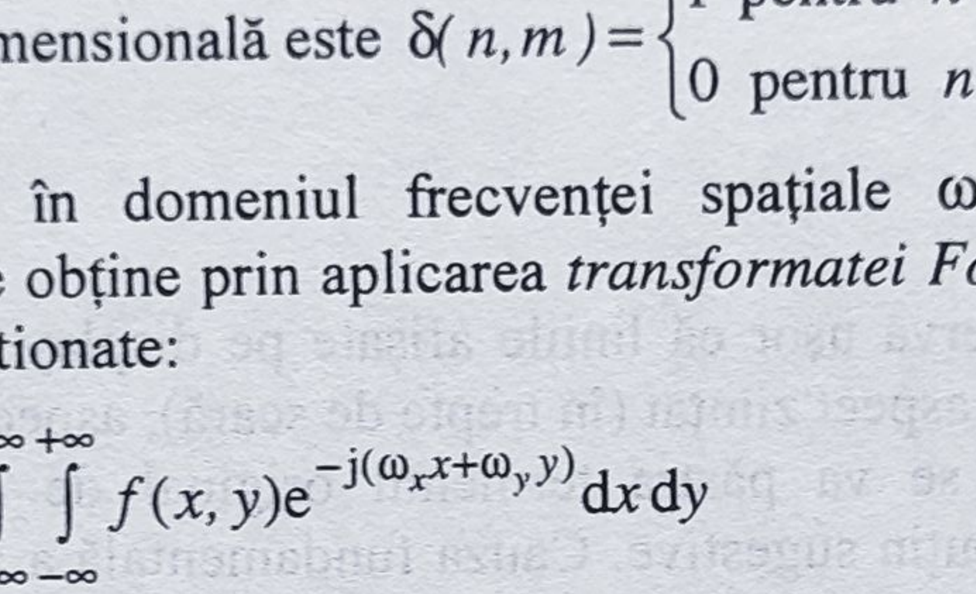


Fig. 9.1 Matricea de eşantionare spaţială cu funcţii delta.

Transformata Fourier bidimensională a funcţiei de eşantionare spaţială este constituită dintr-o matrice infinită de funcţii delta în domeniul frecvenţei spaţiale:

$$S(\omega_x,\omega_y)=\frac{4\pi^2}{\Delta x\Delta y}\sum_{n=-\infty}^{+\infty}\sum_{m=-\infty}^{+\infty}\delta(\omega_x-n\omega_{xs},\omega_y-m\omega_{ys}) \tag{9.5}$$

unde $\omega_{xs}=\frac{2\pi}{\Delta x}$ şi $\omega_{ys}=\frac{2\pi}{\Delta y}$ reprezintă frecvenţele de eşantionare spaţială.

Presupunând că imaginea are o bandă limitată (*band-limited*), astfel că:

$$I(\omega_x, \omega_y) = 0 \quad pentru \left|\omega_x\right| > \omega_{xc} \; si \left|\omega_y\right| > \omega_{yc} \tag{9.6}$$

şi făcând produsul de convoluţie cu matricea de eşantionare, se obţine:

$$F(\omega_x, \omega_y) = \frac{1}{\Delta x \Delta y} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\omega_x - u, \omega_y - v) \cdot \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} \delta(u - n\omega_{xs}, v - m\omega_{ys}) du dv \tag{9.7}$$

Dacă se schimbă ordinea de însumare şi se aplică proprietatea de deplasare a funcţiei delta, se obţine spectrul imaginii eşantionate:

$$F(\omega_x, \omega_y) = \frac{1}{\Delta x \Delta y} \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I(\omega_x - n\omega_{xs}, \omega_y - m\omega_{ys}) \tag{9.8}$$

Reprezentarea grafică a relaţiei (9.8), dată în fig. 9.2(b), evidenţiază faptul că spectrul imaginii eşantionate constă din spectrul imaginii originale care se repetă la infinit în planul frecvenţelor, pe o matrice de rezoluţie ($2\pi/\Delta x$, $2\pi/\Delta y$).

Imaginea se reconstruieşte printr-un *filtru bidimensional trece-jos* $H(\omega_x, \omega_y)$ cu frecvenţele de tăiere egale cu jumătatea frecvenţelor de eşantionare (fig.9.8(c)).

Se poate considera că frecvenţele de eşantionare şi frecvenţele maxime ale imaginii sunt egale pe cele două coordonate: $\omega_{xs} = \omega_{ys} = \omega_s$ şi $\omega_{xc} = \omega_{yc} = \omega_c$. Relaţia dintre frecvenţa de eşantionare (ω_s) şi frecvenţa spaţială maximă a imaginii (frecvenţa de tăîere ω_c) exprimă posibilitatea de refacere a imaginii după eşantionare [Hua75].

Dacă frecvenţa spaţială maximă a imaginii este mai mică sau egală cu jumătate din frecvenţa de eşantionare($\omega_c \leq \omega_s/2$), atunci, prin filtrarea imaginii eşantionate, se obţine imaginea originală nedistorsionată. Intervalul $\omega_s/2$ este numit *limita Nyquist*.

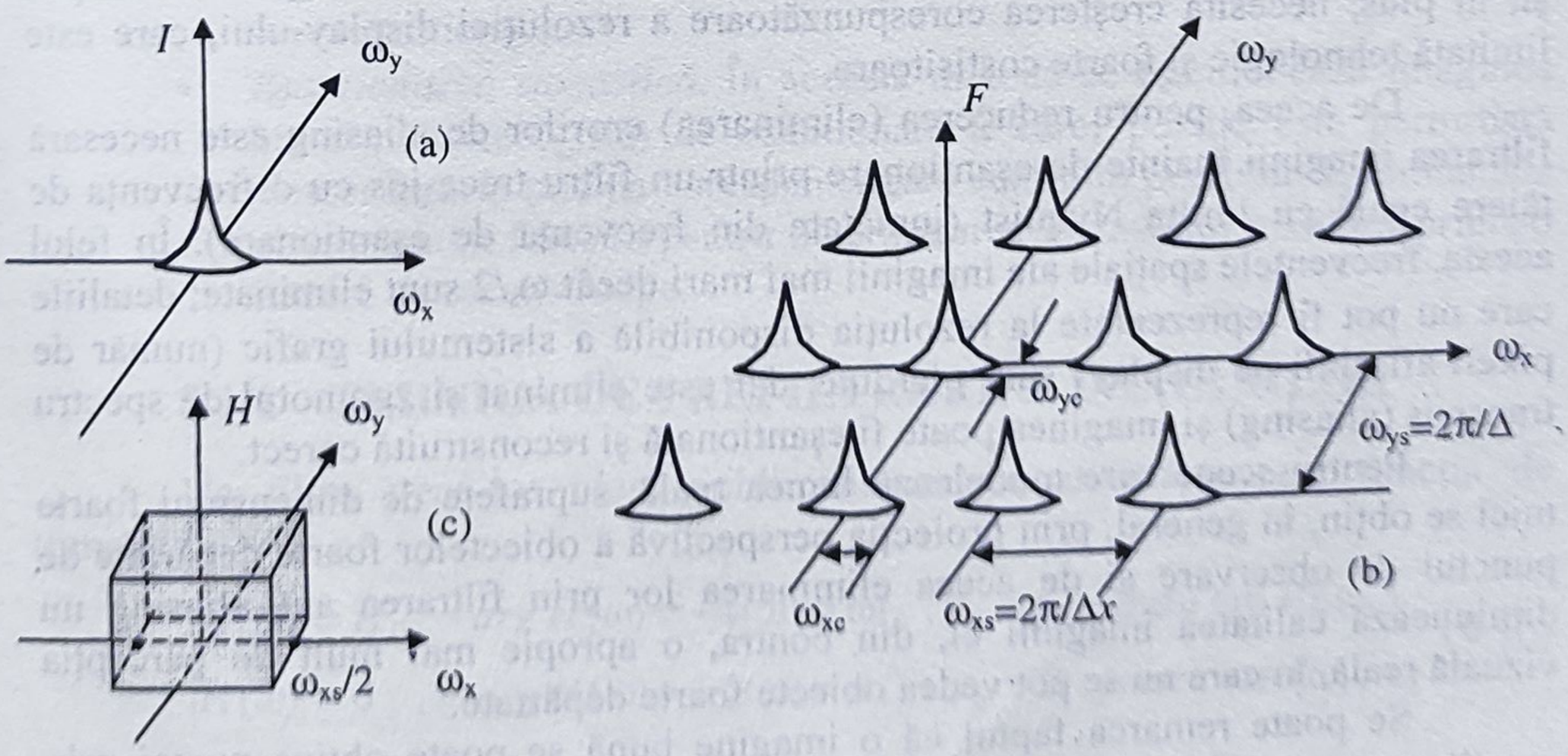


Fig. 9.2 (a) Spectrul imaginii originale. (b) Spectrul imaginii eşantionate. (c) Filtrul de reconstrucţie ideal.

Dacă frecvenţa spaţială maximă a imaginii este mai mare decât jumătate din frecvenţa de eşantionare ($\omega_c > \omega_s/2$), atunci spectrele imaginii eşantionate se suprapun, producând un *zgomot de spectru transpus* cunoscut sub numele de *aliasing*. Această suprapunere de spectru distruge o parte din informaţia imaginii originale: frecvenţele înalte (detalii ale imaginii) se pierd şi apar interferenţe (*aliases*) în regiunea frecvenţelor joase.

Această interpretare este extensia la spaţiul bidimensional a teoremei eşantionării. Conform teoremei eşantionării, o funcţie continuă de o singură variabilă poate fi complet reprezentată printr-o mulţime de eşantioane calculate la intervale egale, dacă intervalul dintre două eşantioane este mai mic decât jumătate din perioada celei mai ridicate frecvenţe componente a funcţiei.

Eşantionarea şi redarea imaginii în grafica pe calculator se realizează prin calculul intensităţii în centrul pixelilor şi asignarea acestei valori întregii zone spaţiale care corespunde unui pixel.

Erorile de aliasing apar numai dacă spectrul frecvenţelor spaţiale ale imaginii conţine valori mai mari decât jumătate din frecvenţa de eşantionare. O metodă de a reduce erorile de aliasing este aceea de a creşte frecvenţa de eşantionare, adică de a creşte rezoluţia imaginii. Această metodă reduce, dar nu elimină zgomotul de aliasing, dat fiind că spectrul frecvenţelor spaţiale ale imaginii se poate extinde către infinit. Acestă situaţie este mai pregnantă în reprezentarea prin proiecţie perspectivă a imaginilor tridimensionale. Din relaţia (4.10), prin care se calculează coordonatele punctelor prin proiecţie perspectivă: $x_N = d\,x_V / g\,z_V$, $y_N = d\,y_V / h\,z_V$ rezultă că, pentru $z_V \to \infty$, $x_N \to 0$, $y_N \to 0$, adică toate punctele aflate de la distanţe foarte mari faţă de punctul de observare se proiectează foarte aproape de centrul ferestrei de proiecţie. Prin proiecţie perspectivă, obiectele aflate la distanţe foarte mari de punctul de observare produc elemente ale imaginii de dimensiuni foarte mici şi deci cu o frecvenţă spaţială care tinde către infinit.

Deci creşterea rezoluţiei imaginii nu elimină zgomotul de spectru transpus şi, în plus, necesită creşterea corespunzătoare a rezoluţiei display-ului, care este limitată tehnologic şi foarte costisitoare.

De aceea, pentru reducerea (eliminarea) erorilor de aliasing este necesară filtrarea imaginii înainte de eşantionare printr-un filtru trece-jos cu o frecvenţa de tăiere egală cu limita Nyquist (jumătate din frecvenţa de eşantionare). În felul acesta, frecvenţele spaţiale ale imaginii mai mari decât $\omega_s/2$ sunt eliminate; detaliile care nu pot fi reprezentate la rezoluţia disponibilă a sistemului grafic (număr de pixeli afişabili pe display) sunt pierdute, dar este eliminat şi zgomotul de spectru transpus (aliasing) şi imaginea poate fi eşantionată şi reconstruită corect.

Pentru scene care modelează lumea reală, suprafeţe de dimensiuni foarte mici se obţin, în general, prin proiecţia perspectivă a obiectelor foarte depărtate de punctul de observare şi de aceea eliminarea lor prin filtrarea anti-aliasing nu diminuează calitatea imaginii ci, din contra, o apropie mai mult de percepţia vizuală reală, în care nu se pot vedea obiecte foarte depărtate.

Se poate remarca faptul că o imagine bună se poate obţine numai prin combinarea celor două metode: creşterea, atât cât este posibil din punct de vedere

tehnologic, a rezoluției display-ului, prin care crește frecvența de eșantionare a imaginii, și filtrarea anti-aliasing, prin care se elimină detaliile cu frecvențe spațiale mai mari decât jumătate din frecvența de eșantionare, precum și zgomotul de spectru transpus (aliasing).

În grafica pe calculator au fost dezvoltate mai multe tehnici de filtrare a imaginilor pentru diminuarea efectelor de aliasing. Aceste tehnici, numite *tehnici de anti-aliasing*, încearcă să implementeze prin calcul cât mai eficient operația de filtrare a imaginilor. În esență, tehnicile de anti-aliasing reprezintă un compromis între simplitatea calculelor (și deci eficiența acestora) și rezultatul, din punct de vedere al percepției vizuale, al filtrării imaginilor.

Filtrarea, care este o multiplicare în domeniul frecvențelor, se poate calcula printr-o convoluție în domeniul spațial prin integrala:

$$f(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(x-u, y-v)\, h(u,v)\, \mathrm{d}u\, \mathrm{d}v \qquad (9.9)$$

unde: $f(x,y)$ este imaginea filtrată;
$i(x,y)$ este imaginea nefiltrată
$h(u,v)$ este funcția pondere a filtrului.

Cele mai cunoscute metode de diminuare a aliasing-ului, care vor fi descrise în continuare, sunt:

- *Prefiltrarea imaginii.* În această metodă imaginea este filtrată cu un filtru trece-jos cu frecvența de tăiere egală cu intervalul Nyquist. Algoritmii dezvoltați în acestă tehnică se concentrează asupra aproximării cât mai bune și eficiente a funcției pondere a filtrului. Cele mai multe din sistemele grafice actuale implementează o variantă a prefiltrării, printr-o anumită aproximare a funcției pondere a filtrului.
- *Supraeșantionarea imaginii* (*postfiltrare*). Acestă metodă implementează convoluția în domeniul discret a unei imagini supraeșantionate.
- *Eșantionarea stocastică.* În această metodă se eșantionează imaginea într-o matrice (grilă) de eșantionare a cărei poziție este perturbată aleator față de poziția corespunzătoare centrului pixelilor. Eșantioanele obținute sunt folosite pentru determinarea intensității pixelilor printr-o filtrare de reconstrucție.

9.1.1 TEHNICA DE PREFILTRARE A IMAGINILOR

Un filtru trece-jos ideal unidimensional se definește prin funcția de transfer:

$$\begin{aligned} &|H(\omega)| = H_0, -arg\, H(\omega) = -\phi(\omega) = \omega t_0\ , \tau_g = t_0 \text{ pentru } \omega \in (0, \omega_t), \\ &|H(\omega)| = 0\quad , -arg\, H(\omega) = -\phi(\omega) = \pi, \quad \tau_g = 0 \text{ pentru } \omega \in (\omega_t, \infty) \end{aligned} \qquad (9.10)$$

Funcția pondere a unui filtru, $h(t)$, este răspunsul filtrului la funcția impuls Dirac. Pentru calculul răspunsului filtrului trece-jos la funcția impuls Dirac, se folosește relația transformatei Fourier inverse:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)\mathrm{e}^{\mathrm{j}\omega t}\,\mathrm{d}t \tag{9.11}$$

Înlocuind expresia lui $H(\omega)$ în relația (9.11) se obține:

$$h(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} H_0\mathrm{e}^{-\mathrm{j}\omega t_0}\mathrm{e}^{\mathrm{j}\omega t}\,\mathrm{d}\omega = \frac{H_0}{\pi}\int_{0}^{\omega_t}\cos\omega(t-t_0)\mathrm{d}\omega \tag{9.12}$$

Efectuând integrala rezultă:

$$h(t) = \frac{H_0}{\pi}\frac{sin\,\omega_t(t-t_0)}{t-t_0} \tag{9.13}$$

Dacă se notează $v_t = \omega_t(t-t_0)$, atunci:

$$h(v_t) = \frac{H_0\omega_t}{\pi}\frac{\sin v_t}{v_t} = \frac{H_0\omega_t}{\pi}\operatorname{sinc} v_t \tag{9.14}$$

Funcția pondere a unui filtru trece-jos unidimensional ideal este reprezentată în fig. 9.3.

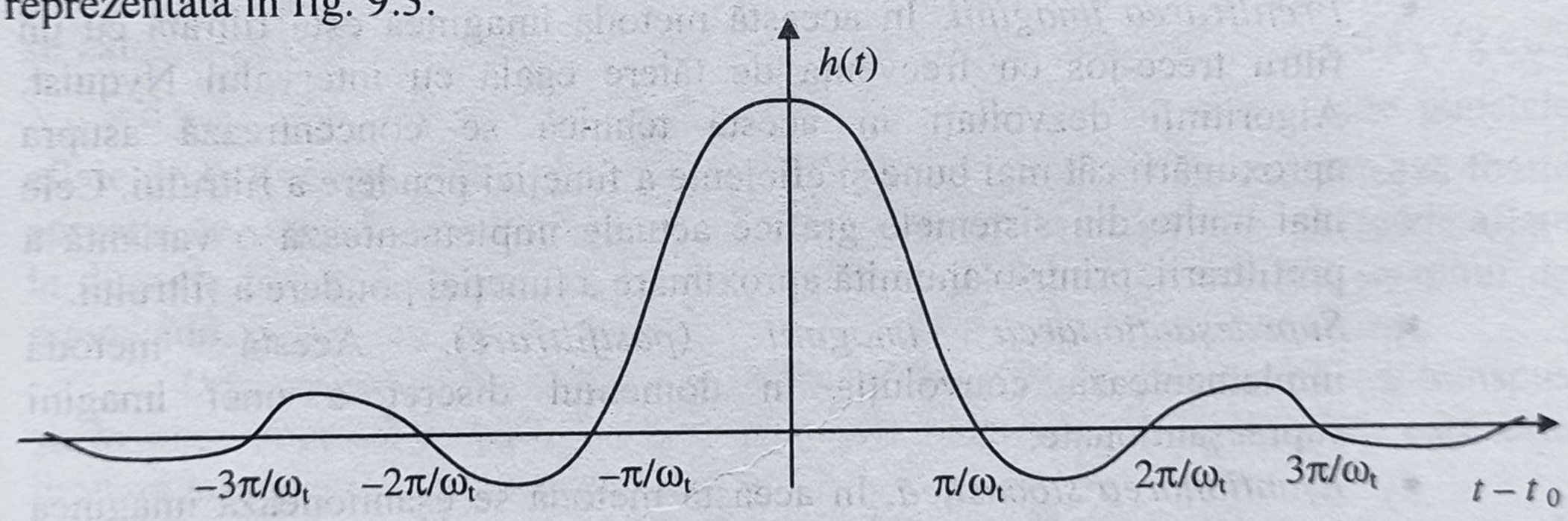


Fig. 9.3 Funcția pondere a unui filtru trece-jos unidimensional ideal.

Pentru filtrarea imaginilor se folosesc filtre bidimensionale, ale căror caracteristici se pot deduce prin extinderea caracteristicilor filtrelor unidimensionale. Problema filtrării imaginilor este de a determina funcția pondere $h(n,m)$ a unui filtru cu caracteristica $H(u,v)$ de filtru trece-jos bidimensional. Această funcție nu se poate determina exact și de aceea se folosesc funcții care aproximează cât mai apropiat și eficient, funcția pondere a unui filtru bidimensional trece-jos.

Tehnicile de prefiltrare pentru diminuarea zgomotului de aliasing se bazează pe aproximarea funcției pondere a filtrului trece jos. În 1981 Crow a testat mai multe funcții de filtrare a imaginilor și a arătat că imaginea reconstruită este cu

atât mai bună cu cât funcția folosită este mai apropiată de funcția pondere a filtrului trece-jos ideal [Crow81]. În fig. 9.4 sunt reprezentate diferite funcții de convoluție în spațiul unidimensional care se pot folosi în prefiltrare. Funcțiile în spațiul bidimensional folosite sunt extensia bidimensională a acestora.

Cea mai puțin costisitoare funcție de convoluție care se folosește în prefiltrarea anti-aliasing este funcția dreptunghiulară pe o distanță interpixel (fig. 9.4 (a)). Această funcție reprezintă o aproximare destul de grosieră a funcției pondere a unui filtru trece-jos ideal. Extensia acestei funcții în spațiul bidimensional este numită fereastră Fourier pe o distanță interpixel, iar integrala de convoluție devine o medie ponderată a intensităților tuturor suprafețelor care acoperă pixelul respectiv.

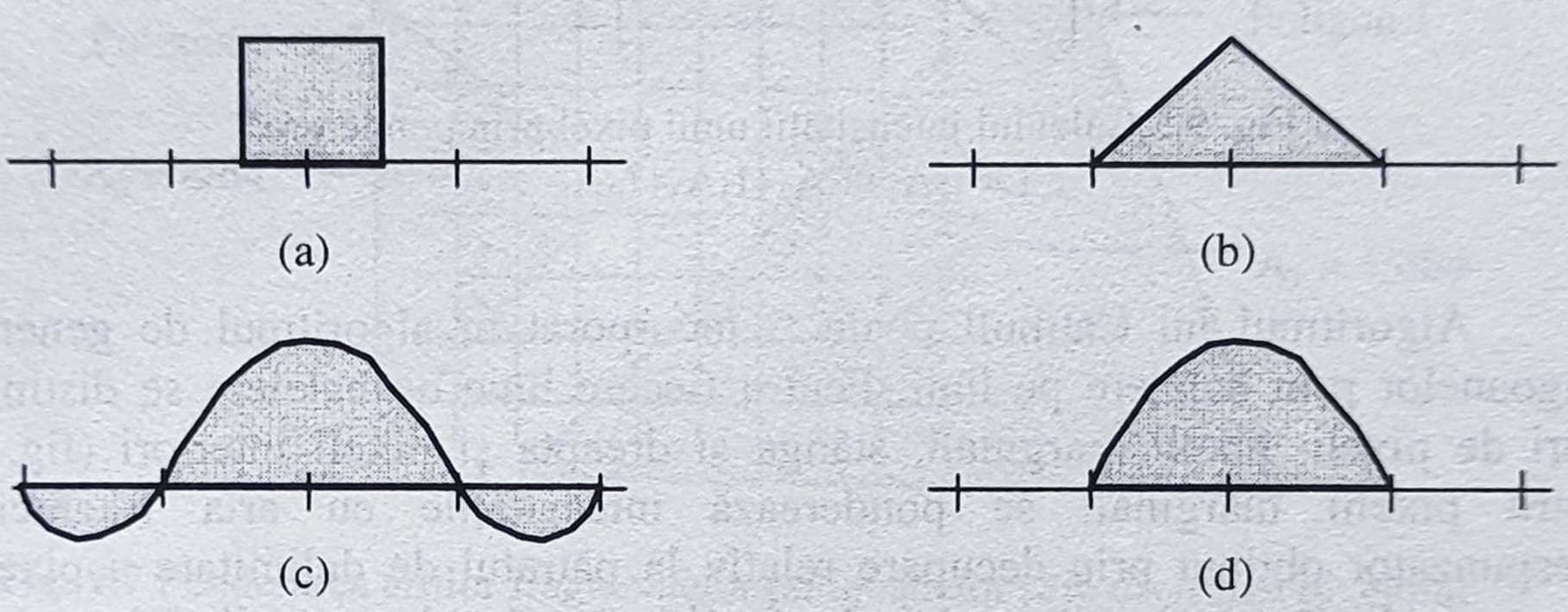


Fig. 9.4 Funcții de convoluție:
(a) dreptunghi pe o distanță interpixel; (b) triunghi peste două distanțe interpixel;
(c) funcția sin x / x pe patru distanțe interpixel;
(d) lobul central al funcției sin x/x pe două distanțe interpixel.

Algoritmul de prefiltrare al lui Catmull. Algoritmul de prefiltrare folosind fereastra Fourier, dezvoltat inițial de Catmull [Cat78], calculează intensitatea fiecărui pixel folosind aria fiecărui *fragment* de suprafață vizibilă în acel pixel ca pondere în însumarea intensităților acestora. Acest mod de calcul este echivalent cu convoluția imaginii cu o funcție pondere a filtrului spațial bidimensional de formă paralelipipedică; valoare intensității calculată prin ponderare este atribuită apoi pixelului. Pentru calculul intensității prin ponderare se efectuează operații la nivel de subpixel în spațiul continuu al imaginii (fig. 9.5).

Pentru calculul intensității unui pixel se execută o operație de decupare a fiecărei suprafețe folosind ca fereastră de decupare pătratul de delimitare a pixelului. Dacă fragmentele mai multor poligoane se suprapun, atunci se sortează în funcție de adâncime și se decupează din nou, unul față de altul, pentru a se decide fragmentele de poligoane vizibile în interiorul pixelului. Fie ariile A_1, A_2, ...A_n ale fragmentelor vizibile în pixel, I_1, I_2, ...I_n intensitățile corespunzătoare acestora și I_B intensitatea de bază a pixelului (setată la ștergerea imaginii). Intensitatea rezultantă a pixelului este:

$$I = I_1A_1 + I_2A_2 + ... + I_nA_n + I_B(1 - A_1 - A_2 ..... - A_n) \quad (9.15)$$

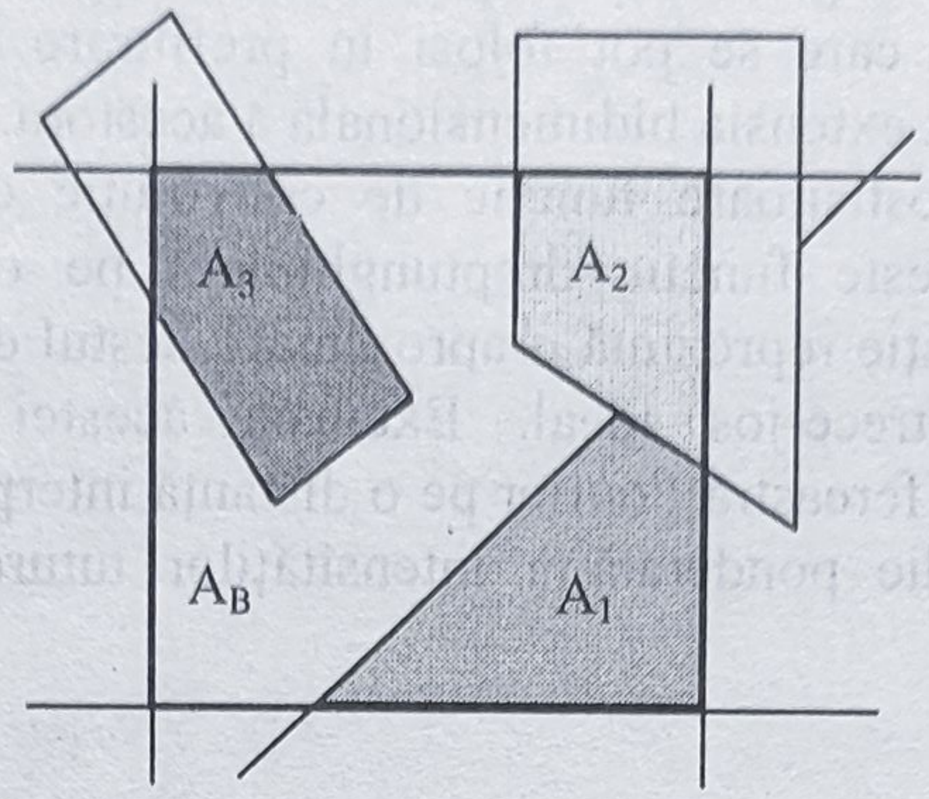


Fig. 9.5 Calculul intensității unui pixel prin ponderare:
$I = I_1A_1+I_2A_2+I_3A_3+I_BA_B$.

Algoritmul lui Catmull poate fi încorporat în algoritmul de generare a poligoanelor prin baleiere pe linii. Pentru fiecare linie de baleiere se disting trei tipuri de pixeli: pixeli marginali, stânga şi dreapta şi pixeli interiori (fig. 9.6). Pentru pixelii marginali se ponderează intensitățile cu aria fragmentului corespunzător obținut prin decupare relativ la pătratul de delimitare a pixelului. Pentru pixelii interiori poligonului (acoperiți complet de un poligon) nu se mai calculează fragmente prin decupare şi se atribuie intensitatea calculată prin interpolare, conform modelului de umbrire folosit.

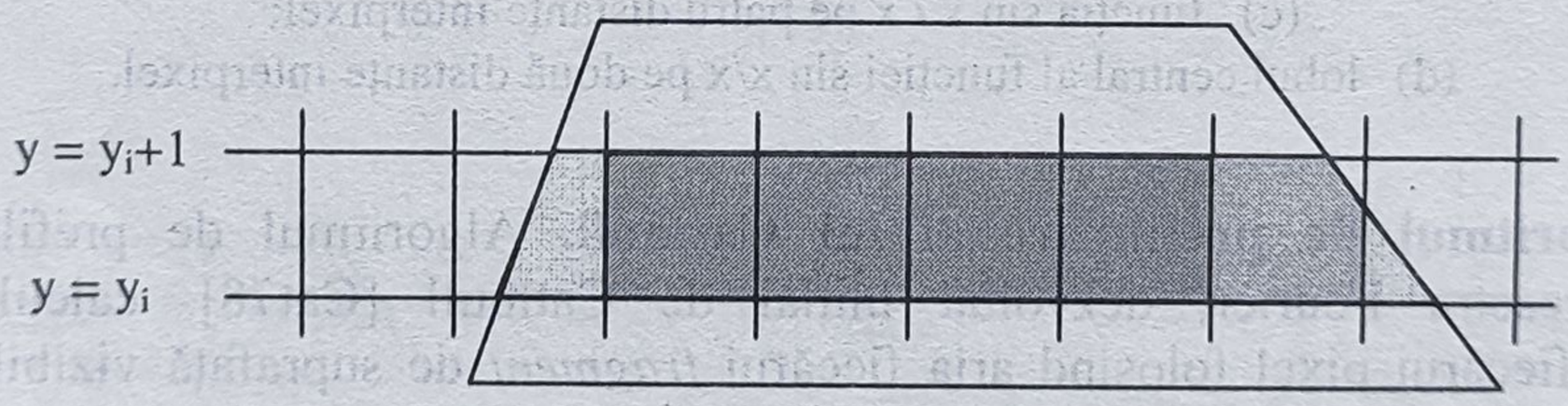


Fig. 9.6 Operația de baleiere pe linii şi prefiltrare anti-aliasing.

Algoritmul original dezvoltat de Catmull este deosebit de costisitor ca timp de execuție. Calculul decupării poligoanelor la limitele pixelului ca şi media ponderată a fragmentelor necesită un mare număr de operații în virgulă flotantă.

Pe baza acestui algoritm s-au dezvoltat ulterior alți algoritmi mai eficienți, care simplifică calculele de ponderare prin aproximarea fragmentele suprafețelor ce acoperă un pixel cu un număr întreg de subpixeli.

Algoritmul de prefiltrare al lui Carpenter. Algoritmul dezvoltat de Carpenter [Car84] foloseşte o tehnică combinată de filtrare anti-aliasing şi Z-buffer, tehnică cunoscută sub numele de A-buffer (*anti-aliased, area-averaged, accumulation buffer*).

Principalul avantaj al acestui algoritm este acela că evită calculele în virgulă flotantă pentru ponderarea ariilor fragmentelor vizibile. Fiecare pixel este divizat într-un număr de k × k subpixeli (fig. 9.7). Aria unui fragment al unui poligon se aproximează cu numărul întreg de subpixeli acoperiți de poligonul respectiv. Un subpixel este considerat acoperit de un poligon dacă este acoperit în proporție mai mare de 0,5 din aria sa. Cu cât numărul de subpixeli în care este divizat pixelul este mai mare, cu atât precizia de calcul a ponderilor intensităților este mai mare şi calitatea imaginii obținute este mai bună.

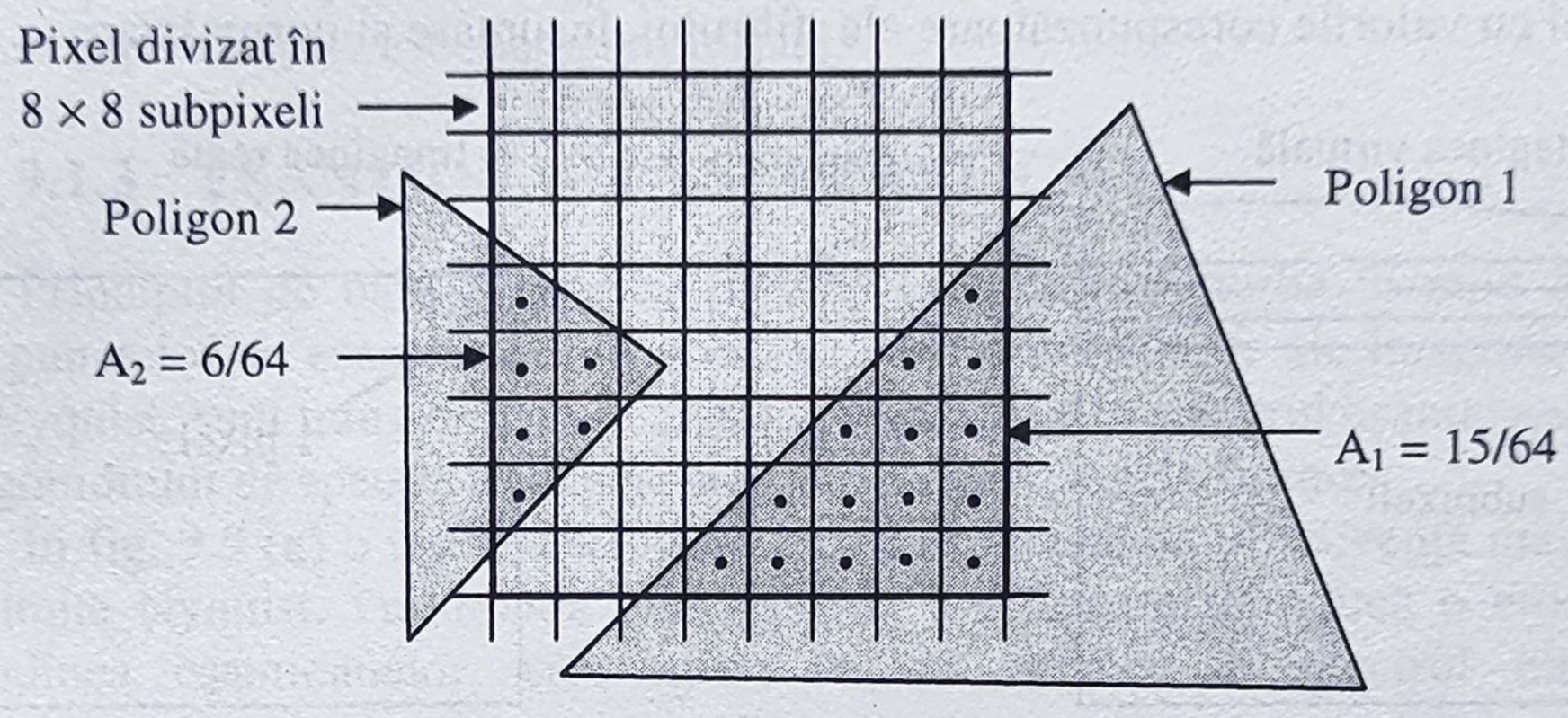


Fig. 9.7 Ponderarea ariilor fragmentelor poligoanelor prin divizarea pixelului.

În majoritatea documentațiilor tehnice calitatea anti-aliasing-ului oferit de un sistem grafic (accelerator grafic) este dată prin numărul k × k de subpixeli. Cu cât numărul de subpixeli este mai mare, cu atât ponderile (ariile fragmentelor) se calculează mai exact şi filtrarea anti-aliasing este mai eficientă.

Detaliile de definire a bufferului de acumulare şi de calcul al ponderilor atribuite fragmentelor suprafețelor sunt complexe, dependente de implementarea specifică şi dificil de descris la modul general. În subcapitolul următor sunt descrise modalitățile de filtrare anti-aliasing oferite de biblioteca OpenGL.

9.1.2 TEHNICA DE SUPRAEŞANTIONARE A IMAGINII

Tehnica de supraeşntionare (sau postfiltrare) pentru reducerea zgomotului de aliasing constă din trei etape de calcul:

(a) Imaginea din spațiul bidimensional continuu este eşantionată printr-o matrice de eşantionare de dimensiune (k×n) × (k×n), unde n × n reprezintă numărul de pixeli care se pot afişa în fereastra de afişare, iar k × k reprezintă factorul de supraeşantionare. În practică acest lucru înseamnă generarea imaginii cu o rezoluție de k × k ori mai mare decât rezoluția displayului.

(b) Imaginea eşantionată este filtrată folosind un filtru trece-jos cu frecvența de tăiere egală cu limita Nyquist a displayului.

(c) Imaginea filtrată este reeşantionată la rezoluția displayului.

Acestă tehnică înseamnă generarea unei imagini virtuale cu o rezoluție mai mare decât rezoluția displayului, urmată de reducerea acestei rezoluții la valoarea rezoluției displayului prin filtrare. Intensitatea fiecărui pixel se determină din valorile unui număr de k × k subpixeli din imaginea virtuală. Acest procedeu este prezentat în fig. 9.8. În acest exemplu, imaginea este generată cu o rezoluție de 3 × 3 ori mai mare decât rezoluția imaginii afișate folosind procedurile cunoscute de conversie prin baleiere pe linii a poligoanelor, combinate cu umbrire și Z-buffer. Grupuri de 3 × 3 subpixeli din imaginea virtuală sunt apoi folosiți pentru generarea intensității fiecărui pixel din imaginea reală prin ponderarea intensității fiecărui subpixel cu valorile corespunzătoare ale filtrului, însumare și normalizare.

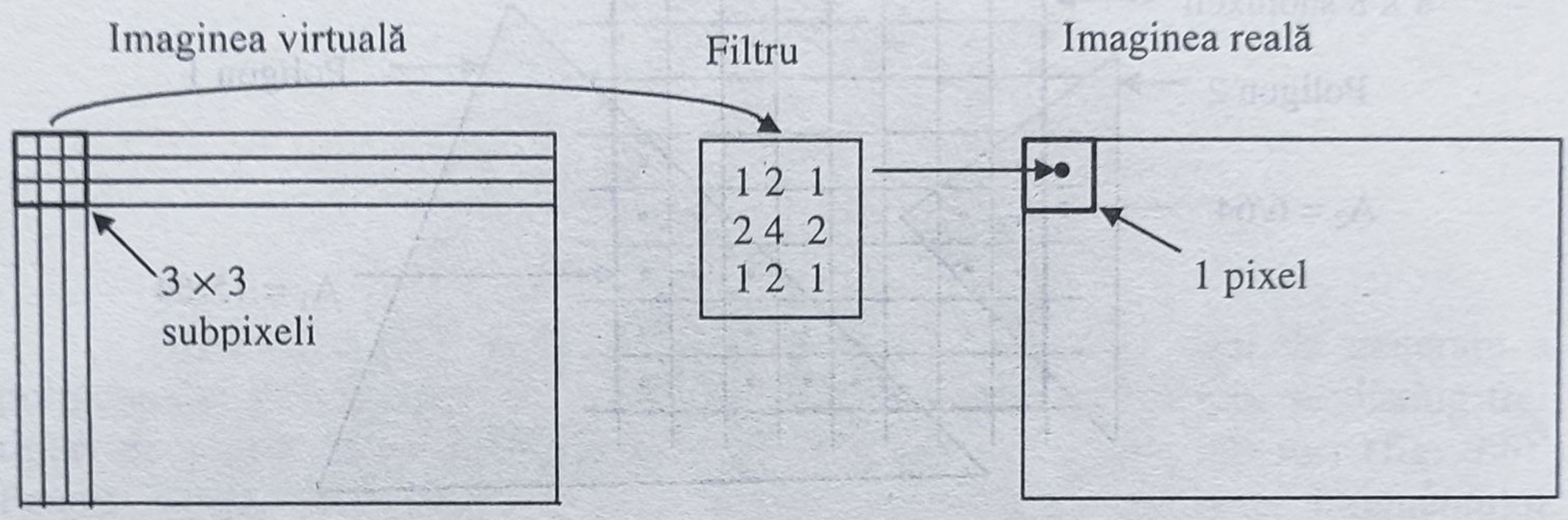


Fig. 9.8 Supraeșantionarea și filtarea imaginii.

Această metodă de ponderare este mai performantă decât medierea corespunzătoare ferestrei Fourier și este cunoscută sub numele de ponderare în fereastra Barlett. Fereastra Barlett poate avea diferite dimensiuni (de exemplu, 3 × 3, 5 × 5 sau 7 × 7) în funcție de factorul de supraeșantionare. Ponderile corespunzătoare ale ferestrei Barlett asigură contribuția subpixelilor la intensitatea finală a pixelului cu atât mai substanțială cu cât sunt mai apropiați de centrul ferestrei (tabelul 9.1).

Tabelul 9.1

Ferestre Barlett folosite în postfiltrarea imaginilor supraeșantionate

3×3	5×5	7×7
1 2 1	1 2 3 2 1	1 2 3 4 3 2 1
2 4 2	2 4 6 4 2	2 4 6 8 6 4 2
1 2 1	3 6 9 6 3	3 6 9 12 9 6 3
	2 4 6 4 2	4 8 12 16 12 8 4
	1 2 3 2 1	3 6 9 12 9 6 3
		2 4 6 8 6 4 2
		1 2 3 4 3 2 1

Operaţia de filtrare pentru reducerea zgomotului de aliasing, definită prin integrala de convoluţie din relaţia (9.9), se execută prin aproximare în tehnica de supraeşantionare. Atunci când se efectuează ponderarea, imaginea este deja eşantionată şi nu este disponibilă funcţia continuă $i(x, y)$, ci numai eşantioane ale acesteia. Îmbunătăţirea aproximării se obţine prin creşterea factorului de supraeşantionare, dar acest lucru este costisitor, atât ca volum de memorie necesar (Z-bufferul creşte şi el cu acelaşi factor de supraeşantionare) cât şi ca timp de execuţie. Această aproximare este asemănătoare cu aproximarea introdusă în tehnica de prefiltrare prin divizarea pixelului în subpixeli, astfel că distincţia între prefiltrare şi postfiltrare este oarecum forţată.

9.1.3 EŞANTIONAREA STOCASTICĂ

Principiul de bază al acestei metode este de a perturba în mod aleator poziţia punctelor de eşantionare. În acest fel, frecvenţele înalte ale imaginii, peste limita Nyquist, sunt transformate în zgomot, care apare în spectrul eşantioanelor în locul zgomotului de spectru transpus, şi poate fi eliminat prin filtrare.

În fig. 9.9 (a) o undă sinusoidală este eşantionată cu o frecvenţă mai mare decât limita Nyquist. Perturbaţia punctului de eşantionare introduce o eroare în amplitudinea eşantioanelor, care apare ca un zgomot în spectrul imaginii eşantionate.

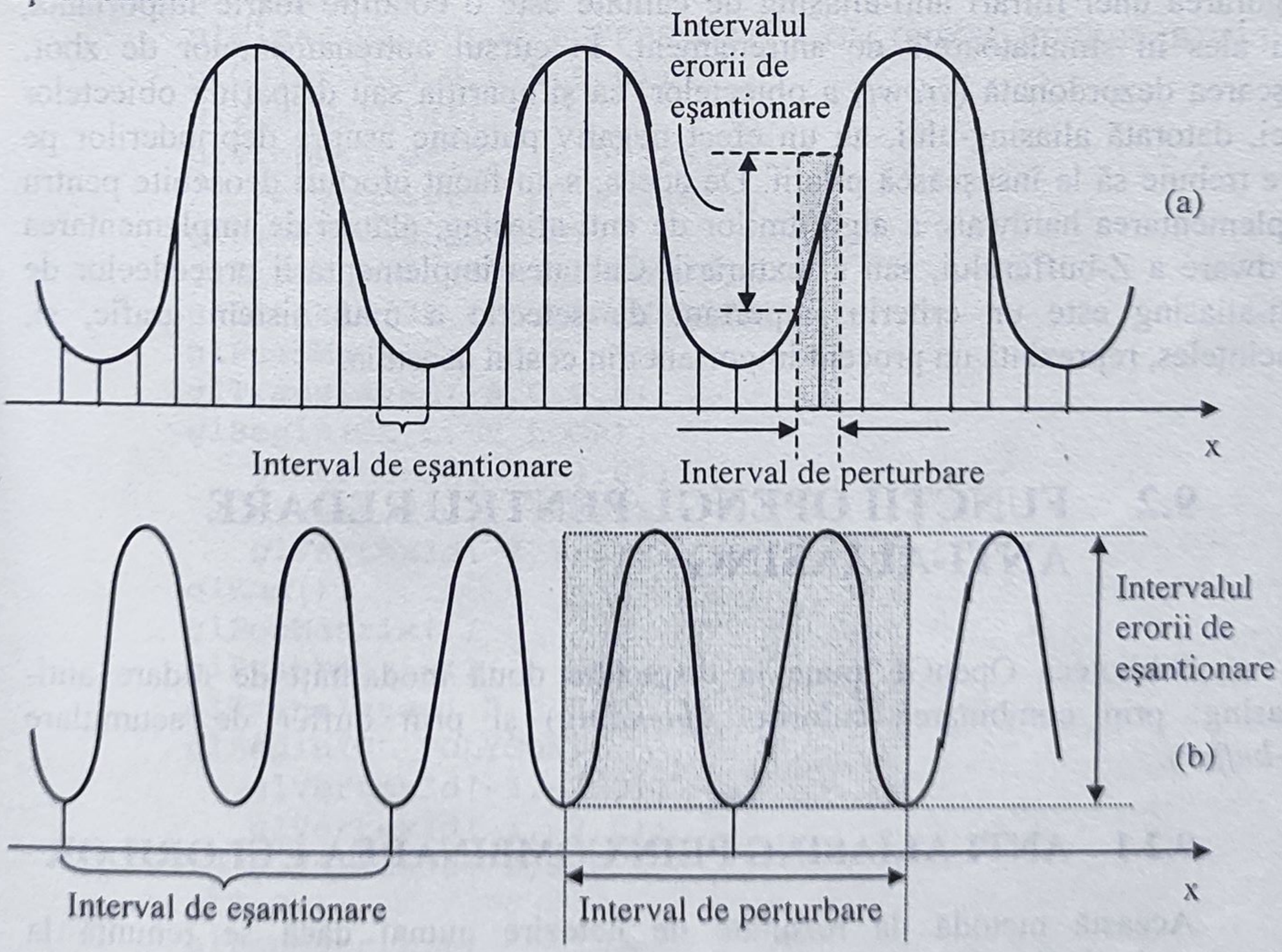


Fig. 9.9 Eşantionarea stocastică a unei unde sinusoidale:
(a) frecvenţa de eşantionare mai mare decât limita Nyquist;
(b) frecvenţa de eşantionare mai mică decât limita Nyquist.

În fig. 9.9 (b) unda sinusoidală este eşantionată cu o frecvenţă mai mică decât limita Nyquist. Perturbaţia punctului de eşantionare se extinde într-un interval egal cu intervalul de eşantionare, iar eroarea de eşantionare rezultată este o eroare aleatoare. În acest fel, eroarea de spectru transpus (aliasing) care ar fi rezultat prin eşantionarea uniformă a acestei unde este înlocuită cu un zgomot aleator, care poate fi eliminat prin filtrare.

În sebcapitolul următor este prezentată modalitatea de implementare a tehnicii de anti-aliasing prin eşantionare stocastică folosind un buffer de acumulare (A-buffer) pentru eliminarea zgomotului aleator rezultat.

Implementarea tehnicilor de anti-aliasing diferă foarte mult de la un sistem grafic la altul. O imagine de calitate (fără zgomot de spectru transpus – aliasing) necesită atât rezoluţii mari ale imaginii, care permit creşterea frecvenţei de eşantionare, cât şi filtrarea imaginii, prin care se limitează spectrul imaginii la o valoare care asigură eşantionarea şi reconstrucţia fără zgomot (sau cu zgomot redus) a imaginii.

Aşa cum se poate observa din descrierile precedente, toţi algoritmii folosiţi pentru diminuarea efectului de aliasing necesită resurse de memorie şi de calcul considerabile şi, cu cât se admit aproximări mai grosiere, cu atât scade calitatea imaginii datorită zgomotului de spectru transpus.

În sistemele grafice folosite în diferite aplicaţii de realitate virtuală, asigurarea unei filtrări anti-aliasing de calitate este o condiţie foarte importantă, mai ales în simulatoarele de antrenament. În cursul antrenamentelor de zbor, mişcarea dezordonată (*crowl*) a obiectelor, ca şi apariţia sau dispariţia obiectelor mici, datorată aliasing-ului, au un efect negativ puternic asupra deprinderilor pe care trebuie să le însuşească piloţii. De aceea, s-au făcut eforturi deosebite pentru implementarea hardware a algoritmilor de anti-aliasing, alături de implementarea hardware a Z-bufferului, sau a texturării. Calitatea implementării procedeelor de anti-aliasing este un criteriu important de selecţie a unui sistem grafic, şi, bineînţeles, reprezintă un procent important din costul acestuia.

9.2 FUNCŢII OPENGL PENTRU REDARE ANTI-ALIASING

Biblioteca OpenGL pune la dispoziţie două modalităţi de redare anti-aliasing: prin combinarea culorilor (*blending*) şi prin buffer de acumulare (*A-buffer*).

9.2.1 ANTI-ALIASING PRIN COMBINAREA CULORILOR

Această metodă dă rezultate de netezire numai dacă se renunţă la mecanismul de Z-buffer (bufferul de adâncime) şi se desenează suprafeţele în ordinea în care sunt transmise bibliotecii. În această situaţie, culoarea unui pixel acoperit parţial de un poligon se obţine prin combinarea culorii poligonului cu

culoarea pixelului, aflată în bufferul de culoare. În modul RGBA, OpenGL multiplică componenta A (*alpha*) a culorii poligonului cu ponderea de acoperire (raportul dintre aria acoperită de fragmentul poligonului şi aria pixelului). Această valoare poate fi folosită pentru ponderarea culorii în fiecare pixel prin combinare cu factorul `GL_SRC_ALPHA` pentru sursă şi factorul `GL_ONE_MINUS_SRC_ALPHA` şi pentru destinaţie. Pentru efectuarea acestor calcule, mai este nesesară validarea anti-aliasing-ului prin apelul funcţiei `glEnable()` cu unul din argumentele `GL_POINT_SMOOTH`, `GL_LINE_SMOOTH`, `GL_POLYGON_SMOOTH` pentru puncte, linii şi, respectiv, poligoane.

■ Exemplul 9.1

Programul din acest exemplu este folosit pentru generarea imaginilor din fig. 9.10 în care se desenează două triunghiuri cu şi fără anti-aliasing. Funcţiile `Init()` şi `Display()` ale programului dezvoltat sub GLUT sunt următoarele:

```
void Init(){
   glClearColor(1.0,1.0,1.0,1.0);
   glLineWidth (1.5);
   glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}
void Display(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3f(0.0,0.0,0.0);
   glPushMatrix();
   glTranslated(0.0,0.0,-25.0);
   // Triunghiuri  cu aliasing
   glDisable(GL_BLEND);
   glDisable(GL_LINE_SMOOTH);
   glDisable(GL_POLYGON_SMOOTH);
   glPushMatrix();
   glTranslated(-8,0,0.0);
   glBegin(GL_LINE_LOOP);
      glVertex3d(-3,-3,0);
      glVertex3d( 3,-2,0);
      glVertex3d( 2,3,0);
   glEnd();
   glPopMatrix();
   glPushMatrix();
   glTranslated(-2,0,0.0);
   glBegin(GL_POLYGON);
      glVertex3d(-3,-3,0);
      glVertex3d( 3,-2,0);
      glVertex3d( 2,3,0);
   glEnd();
   glPopMatrix();
   // Triunghiuri anti-aliasing
   glEnable(GL_BLEND);
   glEnable(GL_LINE_SMOOTH);
```

```
        glEnable(GL_POLYGON_SMOOTH);
        glPushMatrix();
        glTranslated(4,0,0.0);
        glBegin(GL_LINE_LOOP);
           glVertex3d(-3,-3,0);
           glVertex3d( 3,-2,0);
           glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslated(10,0,0.0);
        glBegin(GL_POLYGON);
           glVertex3d(-3,-3,0);
           glVertex3d( 3,-2,0);
           glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();
        glPopMatrix();
        glutSwapBuffers();
}
```

Deoarece anti-aliasing-ul se realizează prin combinarea culorilor, este necesară şi validarea combinării culorilor (`glEnable(GL_BLEND)`) pentru poligoanele cu anti-aliasing.

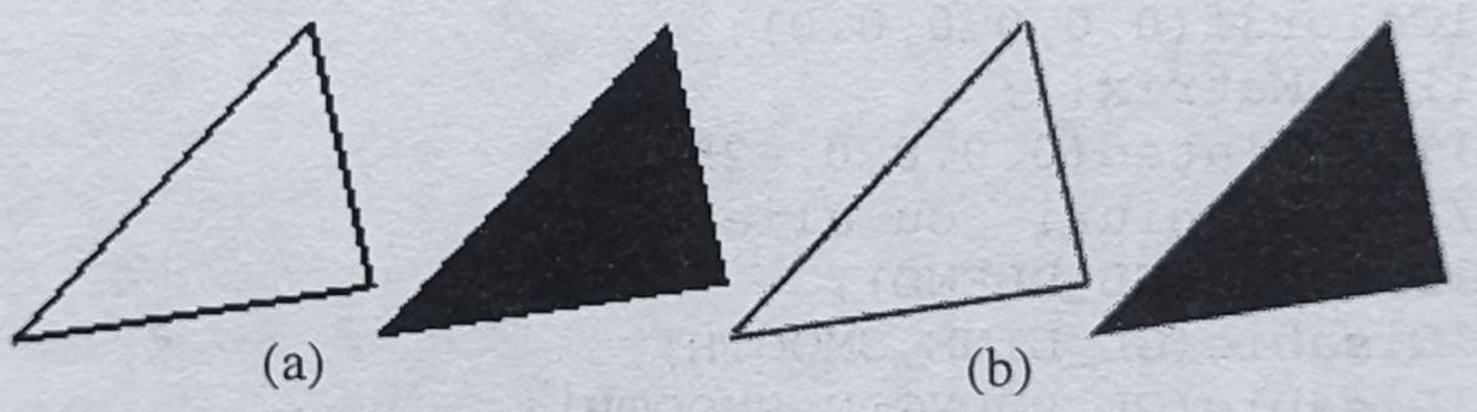


Fig. 9.10 (a) Două poligoane cu aliasing.
(b) Aceleaşi poligoane reprezentate cu anti-aliasing.

Acest mod de filtrare anti-aliasing este foarte simplist şi nu poate fi folosit decât pentru suprafeţe sau linii care nu se suprapun şi pot fi redate fără eliminarea suprafeţelor ascunse, aşa cum a fost situaţia în acest exemplu.

Dacă se redau obiecte tridimensionale complexe sau mai multe obiecte în scenă, este nesesar să fie asigurată eliminarea suprafeţelor ascunse. Combinarea culorilor nu funcţionează corect împreună cu mecanismul de Z-buffer, iar generarea imaginilor fără eliminarea suprafeţelor ascunse este lipsită de realism.

Dacă se invalidează bufferul de adâncime, eliminarea suprafeţelor ascunse trebuie asigurată printr-un alt mecanism, de exemplu prin ordonarea suprafeţelor după adâncime. Dar eliminarea suprafeţelor ascunse prin ordonarea suprafeţelor după adâncime este un procedeu costisitor şi care nu poate fi implementat pentru scene complexe, cu un număr mare de obiecte. Soluţia acceptabilă ca performanţe şi rezultate pentru redarea anti-aliasing a scenelor complexe este folosirea unui buffer de acumulare (A-buffer).

9.2.2 ANTI-ALIASING PRIN ACUMULARE

Biblioteca OpenGL implementează un buffer de acumulare care este folosit atât pentru redarea anti-aliasing a scenelor, cât şi pentru crearea altor efecte vizuale, ca iluminarea cu surse multiple de lumină sau estomparea imaginii unui obiect datorită mişcării acestuia (*motion blur*).

Bufferul de acumulare se defineşte cu aceeaşi dimensiune ca şi bufferul de culoare şi bufferul de adâncime. În fiecare locaţie (x, y) a bufferului de acumulare se memorează culoarea unui pixel, corespunzător pixelului cu aceeaşi adresă (x, y) în bufferul de culoare. Asupra bufferului de acumulare se pot efectua operaţii prin apelul funcţiei:

```
void glAccum(GLenum op, GLfloat value);
```

Parametrul `op` selectează operaţia, iar parametrul `value` este un număr care este folosit în unele operaţii. Operaţiile posibile sunt `GL_ACCUM`, `GL_LOAD`, `GL_RETURN`, `GL_ADD` şi `GL_MULT`.

- `GL_ACCUM` citeşte fiecare pixel din bufferul curent de citire, selectat anterior cu funcţia `glReadBuffer()`, multiplică valorile R,G,B,A ale pixelului cu valoarea `value` şi rezultatul îl adună la valoarea corespunzătoare (cu aceeaşi adresă de pixel) din bufferul de acumulare.
- `GL_LOAD` se execută asemănător cu operaţia `GL_ACCUM`, cu deosebirea că valorile rezultare înlocuiesc valorile existente în bufferul de acumulare.
- `GL_RETURN` preia valorile culorilor fiecărui pixel din bufferul de acumulare, le multiplică cu valoarea `value`, iar rezultatul îl înscrie în buferul de scriere selectat anterior pritr-o funcţie `glDrawBuffer()`.
- `GL_ADD` şi `GL_MULT` adună sau înmulţeşte valoarea componentelor R,G,B,A a fiecărui pixel din bufferul de acumulare cu valorea `value`, rezultatul fiind depus înapoi în bufferul de acumulare. Pentru operaţia `GL_MULT`, `value` se limitează în intervalul [– 1.0, 1.0].

Înainte de începerea fiecărei operaţii de acumulare bufferul de acumulare se şterge prin apelul funcţiei `glClear(GL_ACCUM_BUFFER_BIT)`. Culoarea de ştergere a bufferului de acumulare trebuie să fie (0,0,0,0), şi se setează la iniţializare prin apelul funcţiei `glClearAccum(0.0,0.0,0.0,0.0)`.

Bufferul de acumulare se foloseşte pentru implementarea anti-aliasing-ului prin eşantionare stocastică. Imaginea este eşantionată în poziţii de eşantionare care sunt perturbate aleator faţă de centrul pixelului, pe o distanţă egală cu dimensiunea pixelului. Imaginile succesive obţinute pentru toate poziţiile de eşantionare sunt mediate pentru fiecare componentă de culoare a fiecărui pixel, rezultând imaginea finală, din care a fost eliminat zgomotul aleator datorat perturbării poziţiilor de eşantionare. Fiecare imagine se creează într-un buffer de culoare şi apoi se adaugă ponderat la bufferul de acumulare (iniţial şters) prin funcţia

`glAccum(GL_ACCUM,1/n)`, unde n este numărul de poziții succesive de eşantionare. Imaginea rezultată în bufferul de eşantionare este transferată în bufferul de culoare prin funcția `glAccum(GL_RETURN,1.0)`. La generarea fiecărei imagini corespunzătoare unei poziții de eşantionare se execută testul de adâncime (Z-buffer), deci imaginea finală elimină suprafețele ascunse.

Aşa cum se vede în Planşa 6, efectele de aliasing sunt mult diminuate prin acest procedeu, dar, dacă este implementat soft, timpul de calcul este foarte mare, cel puțin de n ori mai mare decât timpul necesar pentru aceeaşi imagine redată fără anti-aliasing. Implementarea hardware a tehnicii de anti-aliasing este absolut necesară în grafica interactivă. Programul prin care s-a realizat această imagine este dat în exemplul următor.

■ **Exemplul 9.2**

Redarea anti-aliasing a obiectelor cu umbrire şi eliminarea suprafețelor ascunse se poate implementa folosind bufferul de acumulare astfel:

```
#include <GL/glut.h>
#define ACSIZE 16              // numarul de esantionari
static float W = 6;            // dimensiune maxima fereastră
static float WX, WY;           // dimensiuni fereastră
static float N = -10;          // distanta de vizualizare near
static float F = 10;           // distanta de vizualizare far
void Init(void){
GLfloat ambient[] = { 0.4, 0.4, 0.4, 1.0 };
   GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat position[] = { 1.0, 0.2, 1.0, 0.0 };
   GLfloat mat_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat mat_diffuse[] = { 0.9, 0.2, 0.1, 1.0 };
   GLfloat mat_specular[] = { 0.9, 0.6, 0.6, 1.0 };
   GLfloat mat_shininess[] = { 50.0 };
   glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
   glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
   glLightfv(GL_LIGHT0, GL_POSITION, position);
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
   glClearAccum(0.0, 0.0, 0.0, 0.0);
   glClearColor(0.8, 0.8, 0.8, 1.0);
}
GLfloat jitter16[][2] = {
   {0.375, 0.4375}, {0.625, 0.0625},
   {0.875, 0.1875}, {0.125, 0.0625},
```

```
        {0.375, 0.6875}, {0.875, 0.4375},
        {0.625, 0.5625}, {0.375, 0.9375},
        {0.625, 0.3125}, {0.125, 0.5625},
        {0.125, 0.8125}, {0.375, 0.1875},
        {0.875, 0.6875}, {0.875, 0.0625},
        {0.125, 0.3125}, {1.625, 0.8125}
};
void Display(){
    int i;
    glClear(GL_ACCUM_BUFFER_BIT);
    GLint viewport[4];
    glGetIntegerv(GL_VIEWPORT, viewport);
    // Redarea cu anti-aliasing a obiectului
    for (i = 0; i < (ACSIZE); i++) {
      glPushMatrix();
      glTranslatef(jitter16[i][0]*2*WX/viewport[2],
                   jitter16[i][1]*2*WY/viewport[3],0.0);
      glTranslated(1.5,-1,-8);
      glRotatef(15.0, 1.0, 0.0, 0.0);
      glRotatef(20.0, 0.0, 1.0, 0.0);
      glClear(GL_DEPTH_BUFFER_BIT|
              GL_COLOR_BUFFER_BIT);
      glutSolidTeapot(1.0);
      glPopMatrix();
      glAccum(GL_ACCUM, 1.0/(ACSIZE));
    }
    glAccum(GL_RETURN, 1.0);
    // Obiectul redat fara anti-aliasing
    glPushMatrix();
    glTranslated(-1.5,-1,-8);
      glRotatef(15.0, 1.0, 0.0, 0.0);
      glRotatef(20.0, 0.0, 1.0, 0.0);
      glutSolidTeapot(1.0);
  glPopMatrix();
  glFlush();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    if (w <= h){
      WX = W/2;WY = W*(GLfloat) h/(GLfloat) (w*2);
    }
    else{
      WX =W*(GLfloat) w/(GLfloat) (h*2); WY = W/2;
    }
    glOrtho(-WX, WX, -WY, WY, N, F);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|
                        GLUT_DEPTH|GLUT_ALPHA|GLUT_ACCUM);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Acumulare");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

În funcția de inițializare `Init()` se definește sistemul de iluminare (surse de lumină și materiale), se stabilesc culorile de ștergere ale bufferului de culoare și de acumulare și se validează bufferul de adâncime. Unele din comenzile din inițializare nu au mai fost introduse, dacă s-au folosit valorile implicite ale bibliotecii OpenGL. De exemplu, în execuția cu un singur buffer de culoare (stabilită la inițializare), bufferul de citire și scriere curent este implicit bufferul de culoare `GL_FRONT`, și nu s-au mai apelat funcțiile `glReadBuffer()` și `glDrawBuffer()`.

Perturbarea poziției de eșantionare se obține prin perturbarea poziției centrului porții de afișare. În acest exemplu s-a folosit o proiecție ortografică în care transformările inverse de la poarta de afișare în sistemul universal se execută mai simplu. Fereastra este definită ca o fereastră simetrică, de dimensiuni `2*WX`, `2*WY`, calculate în funcția `Reshape()`. Perturbația grilei de eșantionare se obține prin modificarea poziției centrului porții cu valorile aleatoare (ε_x, ε_y) care se citesc din tabelul `jitter`. Acest tabel conține perechi de valori în număr egal cu numărul de acumulări dorite (`ACSIZE`), care sunt poziții aleatoare pe suprafața unui pixel.

Perturbațiile în poarta de afișare se transformă în perturbare a poziției centrului fereastrei de vizualizare prin scalare cu factorii de scalare `2*WX/viewport[2]`, `2*WY/viewport[3]`. Aceste deplasări ale poziției centrului ferestrei de vizualizare sunt echivalente cu deplasări în sens invers ale obiectelelor scenei. În proiecția ortografică, deplasarea obiectelor care conduce la deplasarea centrului porții de afișare și, deci, la perturbarea poziției grilei de eșantionare se obține prin instrucțiunea:

```
glTranslatef(jitter16[i][0]*2*WX/viewport[2],
             jitter16[i][1]*2*WY/viewport[3],0.0);
```

În proiecție perspectivă, deplasarea obiectelor trebuie să fie calculată în funcție de matricea de proiecție perspectivă. Aceste calcule sunt propuse ca exercițiu pentru cititori.

10

TEXTURAREA

Texturarea este o tehnică importantă de creştere a realismului scenelor virtuale, prin care se generează imagini ale obiectelor tridimensionale mult mai interesante şi mai complexe. Prin texturare, se modulează culoarea suprafețelor obiectelor folosind o imagine repetitivă (textura), obținându-se un efect asemănător aceleia de aplicație (lipire) a unui desen pe o suprafață.

În decursul dezvoltării tehnicilor de texturare, au fost propuse diferite modalități de creare a impresiei de realism a suprafețelor, de exemplu prin perturbația normalelor suprafețelor, sau prin variații ale iluminării mediului ambiant al obiectelor. În momentul actual, metoda cea mai larg răspîndită, cu numeroase implementări hardware în sistemele grafice, este metode de modulare a culorii suprafețelor obiectelor folosind imagini de texturi. Texturile care modulează culoarea suprafețelor sporesc informația vizuală prezentată prin sugerarea materialului suprafeței. De exemplu, prin aplicația repetată a imaginii unei cărămizi (obținută prin scanarea unei fotografii a unui perete real), un singur dreptunghi capătă aspectul unui perete întreg (Planşa 7(a)). Desenarea unui astfel de perete folosind câte un poligon pentru fiecare cărămidă este extrem de ineficientă şi imaginea obținută este nerealistă, deoarece fiecare cărămidă ar fi prea netedă şi regulată. De asemenea, prin texturare se poate aplica pe o suprafață o imagine care este practic imposibil să fie generată prin suprafețe poligonale (Planşa 7(b)).

10.1 APLICAȚIA TEXTURILOR

Pentru texturarea obiectelor prin modularea culorii suprafețelor trebuie să se definească următorii parametri:

- Imaginea texturii aplicate.
- Modul de aplicație a texturii pe suprafața unui obiect.
- Modul de diminuare (eliminare) a efectelor de aliasing care apar în aplicarea texturilor (filtrarea texturilor).

Texturile se definesc ca imagini bidimensionale care se aplică suprafețelor obiectelor, sau ca un câmp tridimensional, şi atunci culoarea obiectului se determină prin intersecţia suprafețelor sale cu acest câmp tridimensional al texturii.

10.1.1 APLICAȚIA TEXTURILOR BIDIMENSIONALE

Textura bidimensională este o imagine definită într-un spațiu bidimensional (numit *spațiul texturii*) T(s, t), în intervalul [0,1] pe ambele axe de coordonate s şi t. Textura este alcătuită din elementele componente discrete care sunt dreptunghiuri (sau pătrate) numite *texeli*. Fiecare imagine de textură este definită prin rezoluția ei, dată ca număr de texeli pe cele două coordonate. De exemplu, se folosesc imagini de textură de 256 × 256 texeli, 512 × 512, etc.

Aplicația texturii bidimensionale unui obiect constă din parametrizarea suprafețelor, urmată de transformarea obiectului în sistemul de referință ecran 3D (fig. 10.1). Prin parametrizarea suprafețelor se asociază un punct de coordonate (s, t) în spațiul texturii fiecărui vârf al obiectului modelat (reprezentat în sistemul de referință de modelare). Transformarea completă de aplicație a texturii mai necesită, după parametrizarea suprafețelor, toate transformările geometrice obişnuite, de la sistemul de referință model (spațiul obiect) la sistemul de referință ecran 3D.

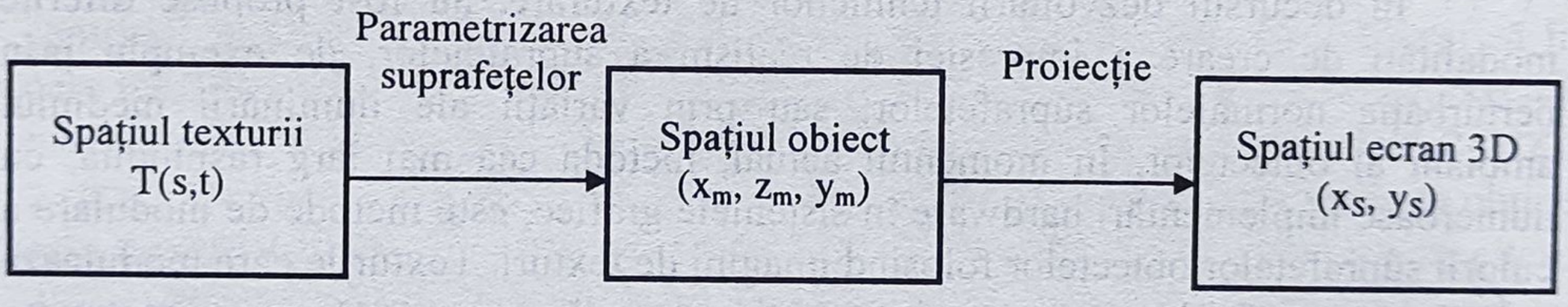


Fig. 10.1 Aplicația texturilor bidimensionale.

Dacă se compun cele două transformări (parametrizarea suprafeței şi transformarea în spațiul ecran 3D) se obține o transformare de aplicație a texturii (*texture mapping*), care transformă puncte din spațiul texturii în spațiul ecran 3D. În fig. 10.2 (a), textura este o tablă de şah, definită în spațiul texturii. Această textură este aplicată unor suprafețe prin definirea coordonatelor fiecărui vârf al suprafeței în spaţiul texturii, iar prin transformarea în spaţiul ecran 3D şi transformarea de rastru, se obține imaginea suprafețelor texturate (fig. 10.2).

Transformarea de parametrizare amplasează imaginea texturii pe un obiect prin asocierea unui punct (s, t) în spațiul texturii (parametrul de texturare), fiecărui punct (x_m, y_m, z_m) al obiectului. Modul cum poate fi colată o suprafață plană (textura) pe un obiect tridimensional nu are o soluție unică şi nu poate fi realizat decât admițând diferite tipuri de erori şi distorsiuni. În implementările practice au fost adoptate mai multe tehnici de aplicație ale texturilor. De exemplu, se poate atribui câte o textură fiecărei fețe a unui obiect (aşa cum este în Planşa 7), unui grup de fețe ale obiectului, sau se poate proiecta aceeaşi textură pe toate fețele obiectului.

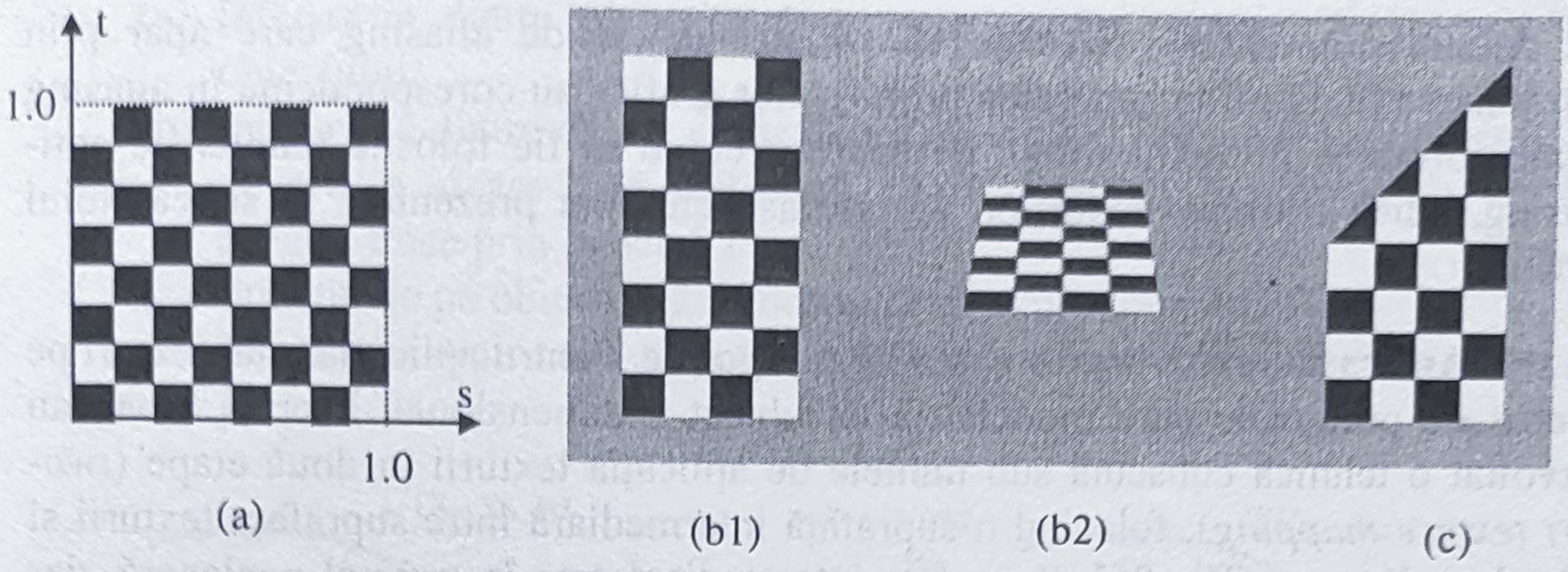


Fig. 10.2 Transformarea de aplicaţie a texturilor:
(a) spaţiul texturii; (b1), (b2), (c) imaginea suprafeţelor texturate.

Aplicaţia texturii pe o suprafaţă plană. La acest mod de aplicaţie a texturii, se definesc coordonatele (s, t) în spaţiul texturii pentru fiecare vârf al suprafeţei.Aceste coordonate sunt calculate o singură dată, la modelarea obiectului, şi sunt memorate împreună cu coordonatele vârfului şi cu normala în vârf.

Dacă $0 \leq s, t \leq 1$, atunci imaginea texturii apare o singură dată (în întregime sau parţial) pe suprafaţa texturată. De exemplu, în fig. 10.2, prima suprafaţă texturată are coodonatele vârfurilor în spaţiul texturii (0.0, 0.0), (0.5, 0.0) (0.5, 1.0), (0.0, 1.0), şi imaginile din fig 10.2(b1) şi (b2) pentru două puncte de observare diferite. Cea de a doua suprafaţă are coordonatele vârfurilor în spaţiul texturii (0.5, 1.0), (1.0, 0.0), (1.0, 1.0), (0.5, 0.5) şi imaginea din fig. 10.2(c).

Dacă coordonatele vârfurilor suprafeţei în spaţiul texturii depăşesc pătratul cu latură egală cu 1 în care este definită imaginea de textură, atunci se poate ca textura să se repete pe suprafaţa texturată. De exemplu, în imaginea din Planşa 7a, faţa verticală a cubului are coordonatele de textură ale vârfurilor (0.0, 0.0), (4.0, 0.0), (4.0, 4.0), (0.0, 4.0). Textura (formată din două "cărămizi") se repetă de patru ori pe suprafaţă (se văd opt cărămizi).

Adresa texelilor corespunzători pixelilor unei primitive geometrice (obţinuţi prin transformarea de rastru) se calculează printr-o *transformare inversă transformării de aplicaţie a texturii.* Un pixel, definit ca un pătrat în spaţiul ecran 3D, este transformat în spaţiul obiect printr-o transformare inversă transformării model-ecran 3D, şi apoi este transformat în spaţiul texturii, printr-o transformare inversă transformării de parametrizare. Deoarece transformarea inversă transformării model-ecran 3D nu este liniară, unui pixel din imagine îi corespunde o suprafaţă în spaţiul texturii care, în general, este un patrulater. Culoarea texelului corespunzător acestui patrulater este atribuită pixelului.

Calculul texelilor în spaţiul texturii corespunzători pixelilor primitivelor geometrice este un proces de eşantionare şi este posibil să apară un zgomot de spectru transpus (aliasing), atunci când frecvenţa de eşantionare este mai mică decât dublul frecvenţei spaţiale maxime a texturii. Prin analogie cu cele prezentate în capitolul precedent, frecvenţa spaţială a texturii este raportul radian/dimensiune, iar frecvenţa spaţială maximă a texturii este dată de inversa dimensiunii celul mai

mic detaliu din textură. Se observă că problemele de aliasing care apar prin eşantionarea imaginii la generarea primitivelor grafice au corespondenţă în aliasing la transformarea de texturare şi, de aceea, trebuie să fie folosite tehnici de anti-aliasing, adică filtrarea texturilor. Aceste aspecte sunt prezentate în subcapitolul următor.

Aplicaţia texturii pe o reţea de poligoane. Pentru aplicaţia unei texturi pe o reţea de poligoane care modelează un obiect tridimensional, Bier şi Sloan au dezvoltat o tehnică cunscută sub numele de aplicaţia texturii în două etape (*two-part texture mapping*), folosind o suprafaţă intermediară între suprafaţa texturii şi grila de poligoane [Bier86]. Suprafaţa intermediară este în general neplanară, dar posedă o funcţie de atribuire analitică prin care textura bidimensională este aplicată cu uşurinţă pe această suprafaţă. După această aplicaţie, corespondenţa dintre un punct al obiectului tridimensional şi un punct în spaţiul texturii se transformă în corespondenţa între spaţiul tridimensional obiect şi spaţiul tridimensional al suprafeţei intermediare. Preţul care se plăteşte în acestă abordare este faptul că apare o dublă distorsiune, deoarece se efectuează două aplicaţii: de la textură la surpafaţa intermediară şi de la suprafaţa intermediară la obiect.

Suprafaţa intermediară poate fi un plan, un cilindru, o sferă sau un cub. Principiul metodei de aplicaţie în două etape a texturii, pentru o suprafaţă intermediară oarecare, poate fi descris astfel:

(a) Aplicaţia de la spaţiul bidimensional al texturii la spaţiul tridimensional al suprafeţei intermediare, numită aplicaţia S: $T(s, t) \rightarrow T'(x_i, y_i, z_i)$.

(b) Aplicaţia de la textura tridimensională la suprafaţa obiectului, numită aplicaţia O: $T'(x_i, y_i, z_i) \rightarrow O(x_m, y_m, z_m)$.

Pentru aplicaţia de la spaţiul bidimensional al texturii la spaţiul tridimensional al suprafeţei intermediare (aplicaţia S) se foloseşte ecuaţia parametrică a suprafeţei intermediare. De exemplu, o suprafaţă cilindrică este definită parametric prin valorile (θ, h). Pentru sferă se foloseşte definiţia parametrică prin unghiurile azimut (θ) şi elevaţie (ϕ). Pentru suprafeţele parametrice (Bézier, B-spline) se foloseşte direct ecuaţia parametrică a acestora.

Atunci când se foloseşte o sferă ca suprafaţă intermediară, apare problema distorsiunii aplicaţiei din planul texturii pe suprafaţa sferei, care creşte către polii sferei. Cubul folosit ca suprafaţă intermediară de texturare este echivalent din punct de vedere topologic cu sfera. O suprafaţă plană se poate aplica pe un cub prin pliere, fără să apară distorsiuni ale texturii, dar textura este decupată la limitele laturilor cubului.

Pentru aplicaţia de la textura tridimensională la suprafaţa obiectului (aplicaţia O) au fost propuse mai multe modalităţi de calcul al corespondenţei dintre punctele pe suprafaţa intermediară de texturare şi punctele obiectului. În fig 10.3 sunt prezentate patru dintre corespondenţele folosite în sistemele grafice. Modalităţile de calcul al acestora sunt:

(a) Intersecţia dintre obiect şi linia care uneşte punctul T' cu centrul obiectului.
(b) Intersecţia dintre obiect şi raza de lumină care porneşte din punctul de observare şi are raza reflectată de suprafaţa obiectului astfel încât aceasta trece prin punctul T'.
(c) Punctul de pe obiect a cărui normală trece prin punctul T'.
(d) Intersecţia dintre obiect şi normala la suprafaţa intermediară care trece punctul T'.

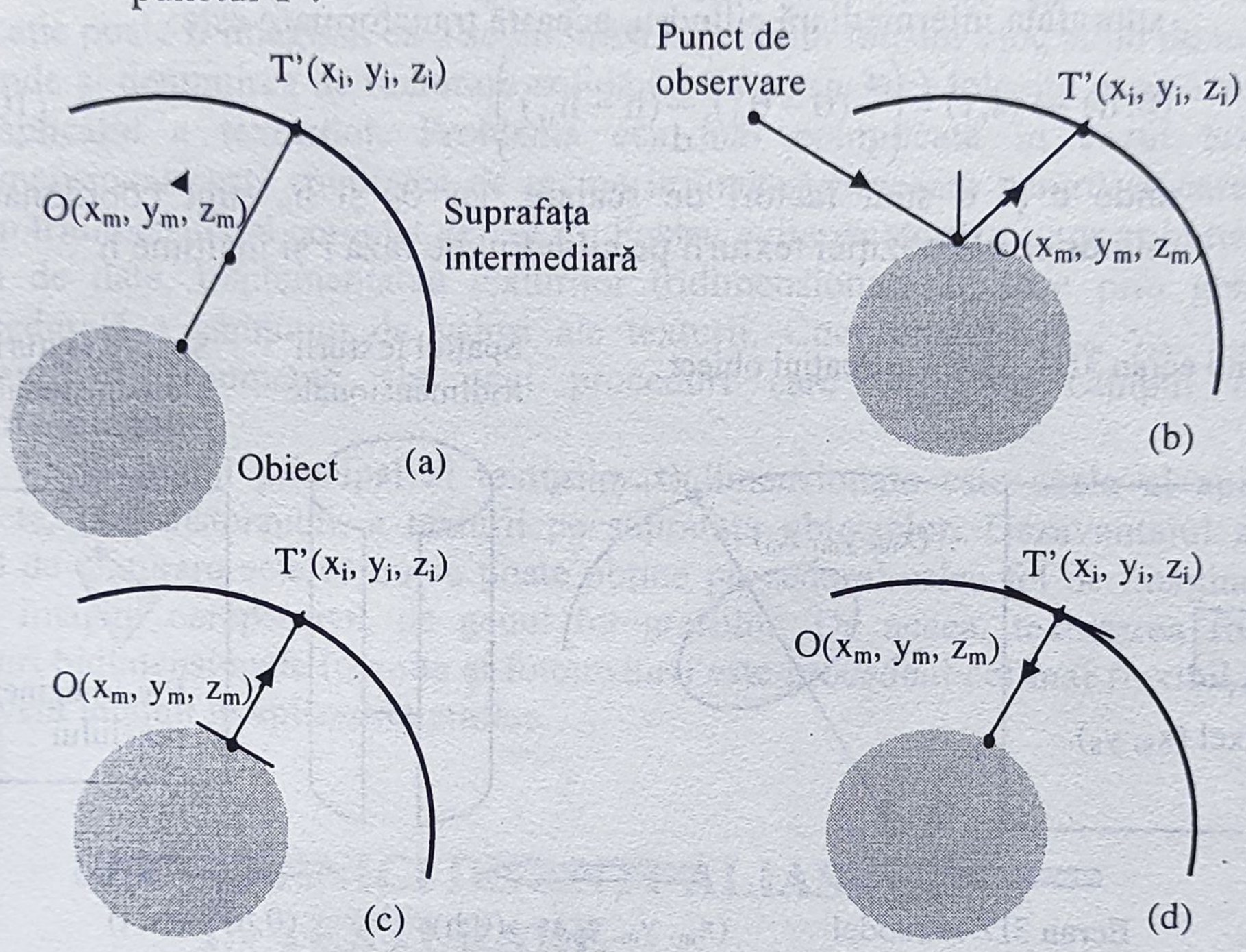


Fig.10.3 Aplicaţii de la suprafaţa intermediară de textură la obiect.

Împreună, cele două aplicaţii, fiecare cu câte patru posibilităţi, dau 16 combinaţii posibile de aplicaţii textură bidimensională-obiect.

Transformarea inversă aplicaţiei texturii (numită transformare de texturare), se calculează pentru fiecare pixel din spaţiul ecran 3D şi constă din trei etape (fig. 10.4):

(1) Transformarea inversă transformărilor modelare-observare-proiecţie, prin care se trece de la coordonatele vârfurilor fiecărui pixel din spaţiul ecran 3D, la coordonatele (x_m, y_m, z_m) a patru puncte pe suprafaţa obiectului (în spaţiul obiect).
(2) Transformarea inversă aplicaţiei O, prin care se calculează coordonatele punctelor pe suprafaţa intermediară de texturare (cilindru, în figura de mai sus), corespunzătoare punctelor în spaţiul obiect. Dacă se consideră cele patru puncte (vârfuri) ale pixelului, se obţine imaginea acestuia pe suprafaţa intermediară de texturare (spaţiul

texturii tridimensionale). Pentru suprafața intermediară cilindru, această transformare este:

$$(x_m, y_m, z_m) \rightarrow (\theta, h) = \left(\frac{1}{\tan(y_m / z_m)}, z_m \right) \tag{10.1}$$

(3) Transformarea inversă aplicației S, prin care se calculează coordonatele (s, t) ale punctelor în spațiul texturii bidimensionale, corespunzătoare punctelor din spațiul texturii tridimensionale. Pentru suprafața intermediară cilindru, această transformare este:

$$(\theta, h) \rightarrow (s, t) = \left(\frac{r}{c}(\theta - \theta_0), \frac{1}{d}(h - h_0) \right) \tag{10.2}$$

unde c şi d sunt factori de scalare, iar θ_0 şi h_0 sunt coordonatele cilindrice ale poziției texturii pe cilindrul de rază r şi înălțime h.

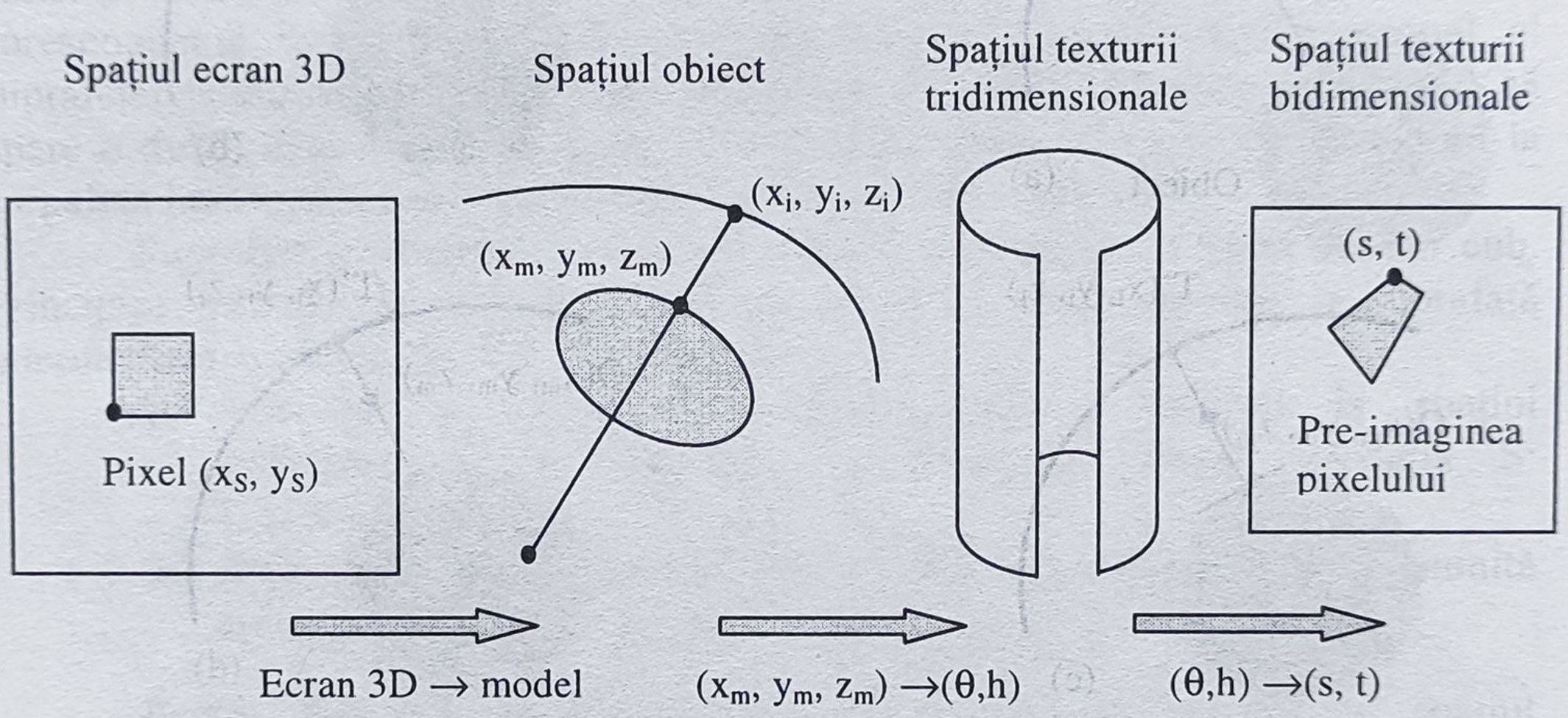


Fig. 10.4 Transformarea de texturare.

Transformarea de parametrizare, adică atribuirea coordonatelor de texturare fiecărui vârf al unui obiect se face, în general, în procesul de modelare a obiectelor. Modelul unui obiect va conține atunci toate informațiile necesare generării imaginii lui: coordonatele vârfurilor, normalele, materialul (culoarea) suprafeței, parametrii (coordonatele) de texturare. Modelarea este un proces off-line, care se efectuează înainte de generarea interactivă a imaginii obiectelor şi poate folosi metode laborioase de parametrizare, dat fiind că nu sunt impuse cerințe de execuție în timp real.

Transformarea de texturare se efectuează în cursul generării interactive a imaginilor. Toate aceste calcule, prin care se prelucrează texelii din imaginea de textură pentru calculul culorii pixelilor imaginii, trebuie executate pentru fiecare pixel al fiecărei suprafațe vizibile a obiectelor din scena virtuală, în fiecare cadru al imaginii. Din această cauză, transformarea de texturare trebuie executată cât mai eficient, iar răspuns în timp real nu se poate obține decât prin implementarea hardware a acesteia.

10.1.2 APLICAȚIA TEXTURILOR TRIDIMENSIONALE

Problema aplicației texturilor este mult mai direct rezolvată pentru texturile tridimensionale. Se poate imagina o textură tridimensională ca un câmp tridimensional continuu de valori în întreg domeniul în care este definit un obiect. Dacă se ignoră problemele de scalare între dimensiunea obiectului şi dimensiunea texturii, atunci unui punct (x_m, y_m, z_m) de pe suprafața unui obiect îi corespunde un parametru de textură dat de aplicația identitate $T(x_m, y_m, z_m)$. Acest mod de aplicație poate fi imaginat ca o sculptură a obiectului într-un bloc solid de material, de unde şi denumirea de texturare solidă (*solid texturing*) folosită pentru acest tip de aplicație a texturilor. Problema cea mai complicată în cazul texturilor tridimensionale este cantitatea de memorie imensă necesară pentru stocarea unui câmp tridimensional complet de valori. Pentru evitarea stocării unor volume foarte mari de date, implementarea texturilor tridimensionale se face prin generarea procedurală a câmpului de valori ale texturii. Coordonatele (x_m, y_m, z_m) sunt folosite ca argumente ale unei proceduri care defineşte câmpul texturii tridimensionale.

Avantajul principal al texturilor tridimensionale este acela al aplicației directe, fără distorsiuni, a texturii pe suprafața obiectelor. Dezavantajul acestui mod de texturare este că nu se poate obține procedural orice fel de imagine, mai ales imagini nerepetitive, de genul fotografiilor. De aceea, texturarea folosind texturi bidimensionale (numite şi fototexturi) este procedeul cel mai flexibil şi mai frecvent întâlnit în aplicațiile grafice.

10.2 TEHNICI DE ANTI-ALIASING ÎN TEXTURARE

Fenomenul de aliasing care apare în texturare are cauza în eşantionarea imaginii de textură. Transformarea din spațiul ecran 3D în spațiul texturii bidimensionale generează o suprafață în formă de patrulater corespunzătoare fiecărui pixel, care este numită *pre-imaginea pixelului*. Pentru simplificarea prezentării se presupune că pre-imaginea pixelului este un pătrat în spațiul texturii (fig. 10.5).

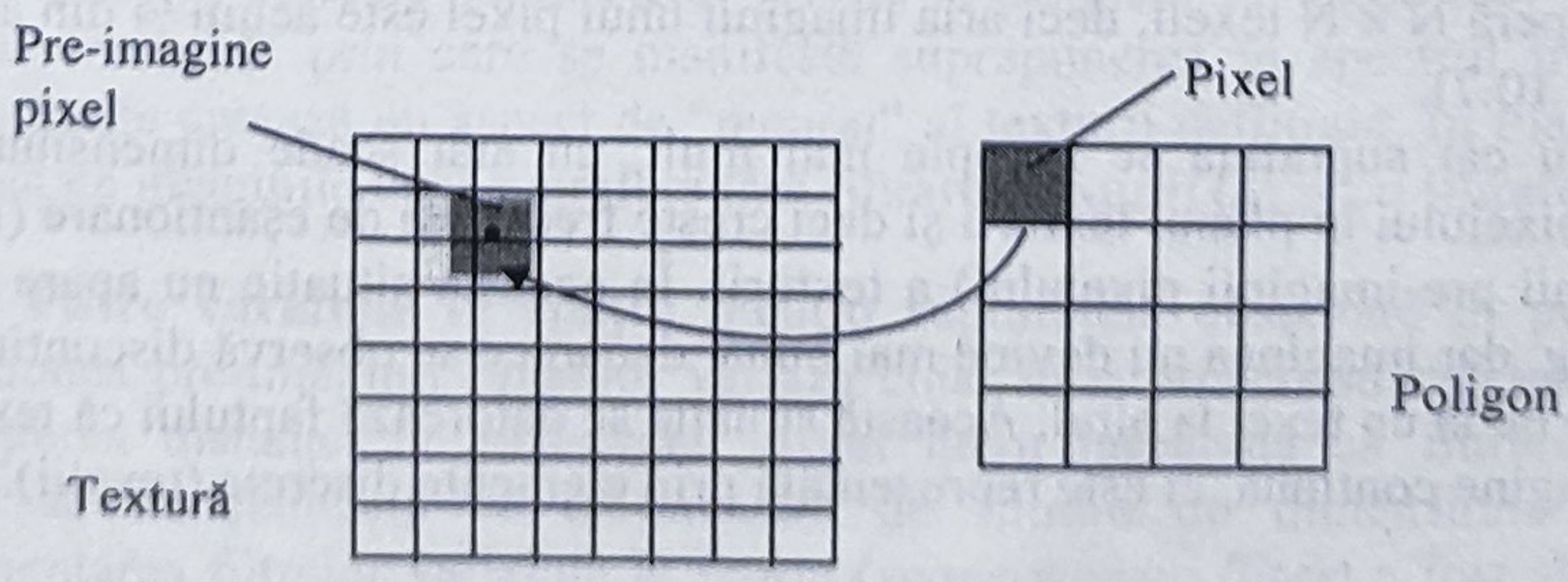


Fig. 10.5 Pre-imaginea unui pixel în spațiul texturii.

Pentru înţelegerea tehnicii de filtrare a texturii, se poate face o analogie directă cu tehnica de prefiltrare anti-aliasing, prezentată în capitolul 9. Imaginea texturii este eşantionată cu o frecvenţă de eşantionare dată de inversul dimensiunii pre-imaginii pixelului în planul texturii. Pentru eliminarea zgomotului de spectru transpus, se foloseşte un filtru trece-jos, aproximat cu fereastra Fourier pe o distanţă inter-pre-imagine pixel, iar integrala de convoluţie este o medie ponderată a tuturor texelilor acoperiţi de pre-imaginea pixelului. În fig. 10.5, culoarea care se atribuie pixelului marcat se obţine prin medierea ponderată a intensităţii culorilor celor patru texeli acoperiţi parţial de pre-imaginea pixelului.

Dacă nu se efectuează filtrarea texturii, atunci se atribuie pixelului culoarea texelului cel mai apropiat de centrul pre-imaginii acestuia. Probelemele de aliasing care apar în această situaţie sunt foarte grave, mai ales pentru imaginile suprafeţelor privite în perspectivă, în care variază mult dimensiunea pre-imaginii pixelilor.

Aria pre-imaginii unui pixel în planul texturii depinde de poziţia suprafeţei faţă de punctul şi direcţia de observare. Se presupune o suprafaţă pătrată cu parametrii texturii (0,0), (1,0), (1,1), (0,1), perpendiculară pe direcţia de observare, la o distanţă D astfel că imaginea ei pe ecran este un pătrat de $N \times N$ pixeli. Se mai presupune, de asemenea, că textura folosită are rezoluţia de $N \times N$ texeli. În această situaţie, aria pre-imaginii unui pixel în spaţiul texturii este egală un aria unui texel (fig. 10.6).

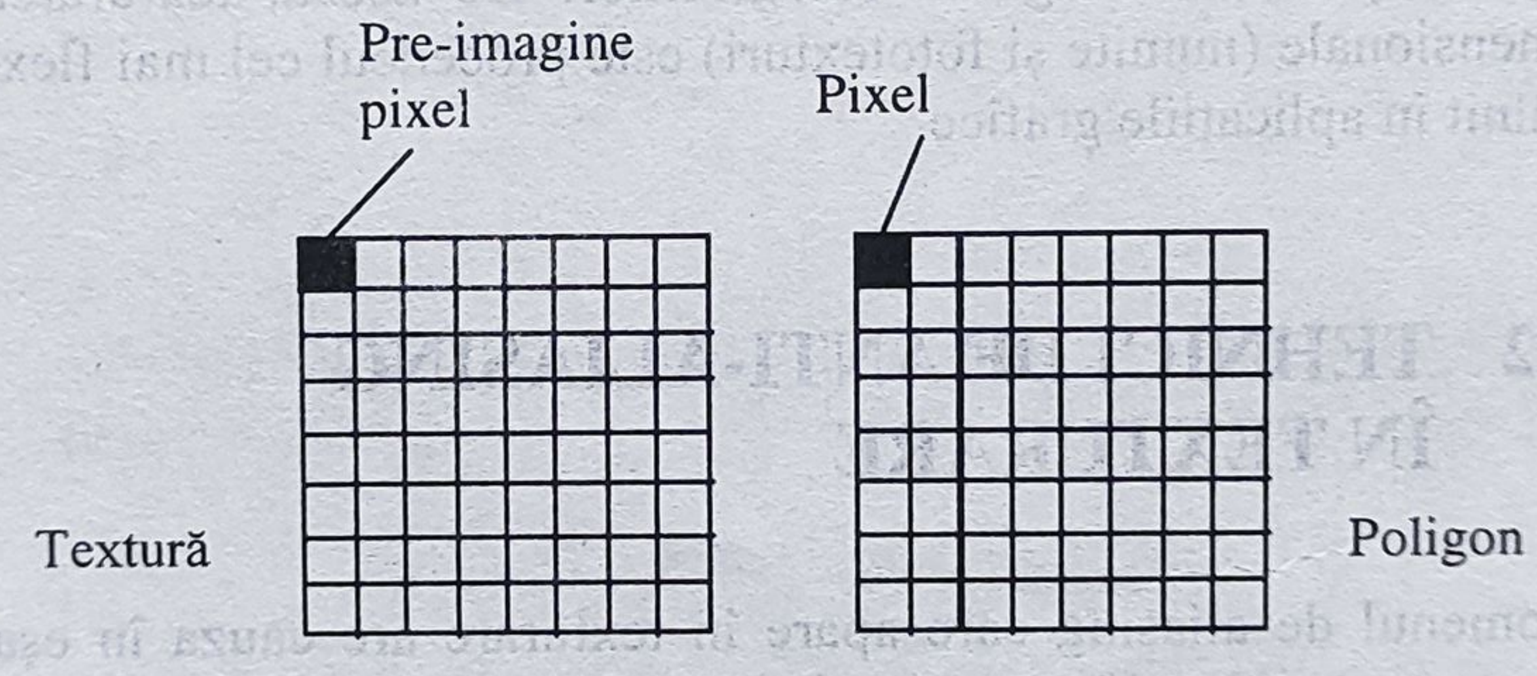


Fig. 10.6 Pixel cu pre-imagine cu aria egală cu aria unui texel.

Dacă suprafaţa se apropie de punctul de observare la distanţă D/2, atunci imaginea suprafeţei pe ecran este de $2N \times 2N$ pixeli şi imaginea celor $2N \times 2N$ pixeli acoperă $N \times N$ texeli, deci aria imaginii unui pixel este acum ¼ din aria unui texel (fig. 10.7).

Cu cât suprafaţa se apropie mai mult, cu atât scade dimensiunea pre-imaginii pixelului în planul texturii şi deci creşte frecvenţa de eşantionare (inversul dimensiunii pre-imaginii pixelului) a texturii. În această situaţie nu apare zgomot de aliasing, dar imaginea nu devine mai bună, deoarece se observă discontinuităţile de trecere de la un texel la altul. Această situaţie se datorează faptului că textura nu este o imagine continuă, ci este reprezentată prin elemente discrete (texelii).

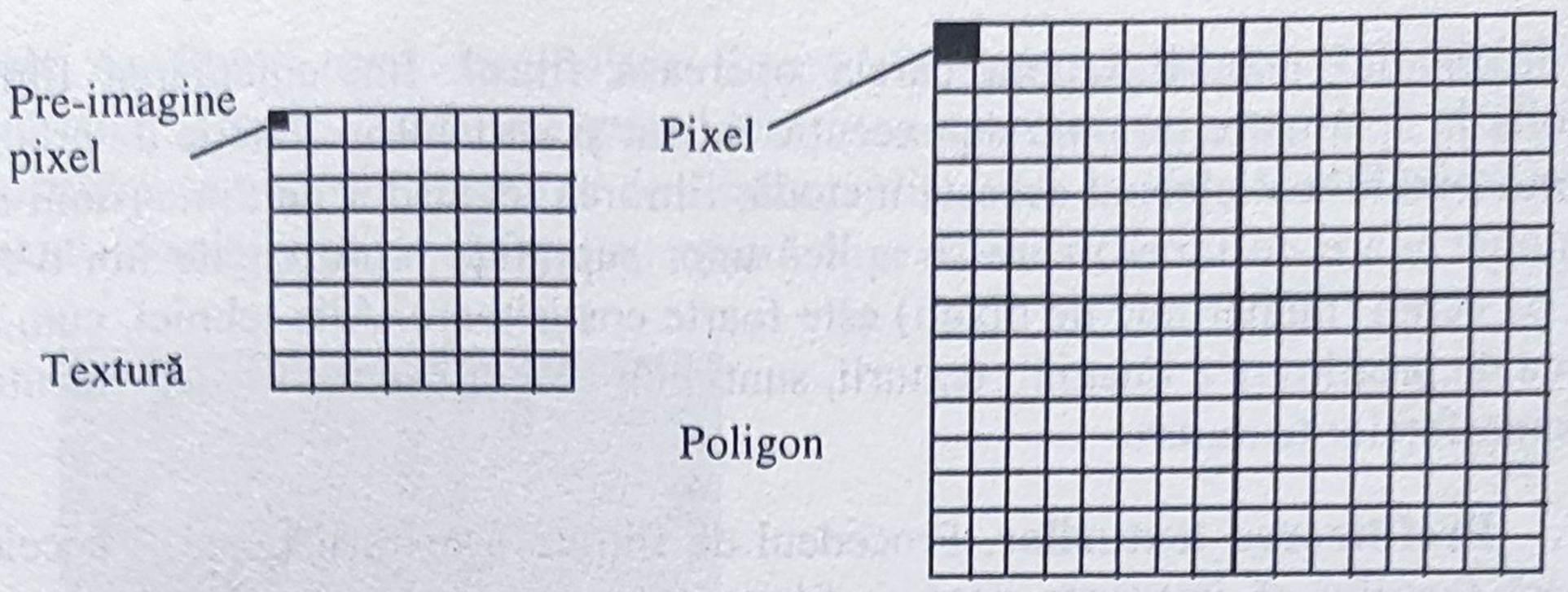


Fig. 10.7 Pixel cu pre-imagine cu arie mai mică decât aria texelului.

Dacă suprafața se depărtează de punctul de observare, de exemplu la distanța 2D, atunci suprafața are dimensiunea de (N/2) × (N/2) pixeli pe ecran este şi imaginea acestora acoperă N × N texeli, deci aria pre-imaginii unui pixel este de patru ori mai mare decât aria unui texel (fig. 10.8).

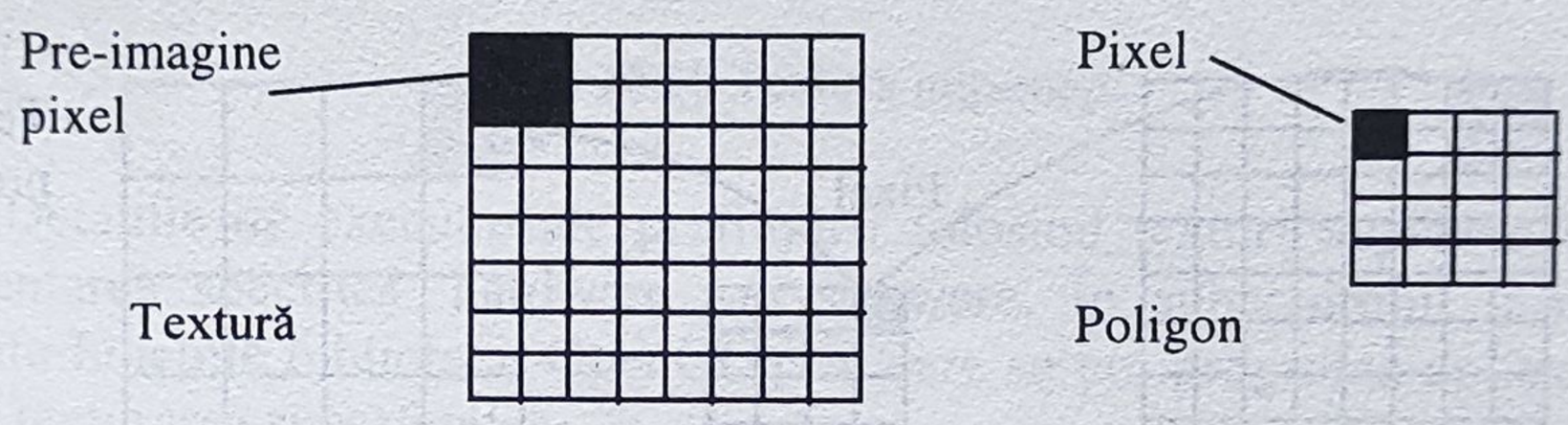


Fig. 10.8 Pixel cu pre-imagine cu aria mai mare decât aria unui texel.

Cu cât suprafața se depărtează mai mult, cu atât creşte dimensiunea pre-imaginii pixelului în planul texturii şi deci scade frecvența de eşantionare a texturii (inversul dimensiunii pre-imaginii pixelului). În această situație apare zgomot de aliasing, dacă nu se efectuează filtrarea imaginii texturii. Filtrarea se execută într-o fereastră Fourier pe dimensiunea pre-imaginii unui pixel în planul texturii, şi se efectuează prin medierea ponderată a intensităților tuturor texelilor acoperiți de pre-imaginea pixelului.

Efectul de aliasing al texturii este deosebit de supărător, mai ales pentru texturi repetitive aplicate suprafețelor privite în perspectivă. Zgomotul în spectrul de joasă frecvență prin care se manifestă suprapunerea în spectrul frecvențelor spaţiale înalte creează un aspect de "moaraj" al texturii nefiltrate. În Planşa 8 sunt reprezentate imaginile unei suprafețe fără filtrarea texturii (a) şi cu filtrare (b).

Filtre variabile în spațiu. Pentru suprafețele observate în perspectivă, dimensiunea pre-imaginii pixelilor variază chiar în cuprinsul suprafeței, crescând cu creşterea distanței de observare. Acest lucru înseamnă că filtrarea texturii trebuie să fie realizată cu o fereastră de filtrare de dimensiune variabilă. Implementarea filtrelor variabile în spațiu (*space-variant filter*) a fost abordată de mai mulți cercetători, folosindu-se aproximarea cu un patrulater sau cu o elipsă a

ariei în spațiul texturii asupra căreia operează filtrul. Implementarea filtrelor variabile în spațiu are un timp de execuție ridicat şi variabil în funcție de aria pre-imaginii pixelilor. Folosind această metodă, filtrarea texturilor de dimensiuni mari (cu număr mare de texeli) care se aplică unor suprafețe reprezentate într-o zonă mică pe ecran (număr mic de pixeli) este foarte costisitoare. Alte tehnici, cum este tehnica de prefiltrare a imaginii texturii, sunt mult mai eficiente şi independente de aria suprafețelor texturare.

Prefiltrarea texturilor. Procedeul de filtrare a texturii poate fi accelerat datorită faptului că imaginea care se filtrează (textura) este cunoscută şi se pot construi imagini prefiltrate ale acesteia. În loc să se efectueze medierea pe suprafața mai multor texeli atunci când pre-imaginea pixelului acoperă un număr mare de texeli, se poate folosi o imagine de textură prefiltrată, cu rezoluție mai scăzută, în care culoarea fiecărui nou texel reprezintă media culorilor texelilor din textura originală (fig. 10.9). Prin prefiltrarea texturii se reduce frecvența spațială maximă a texturii, în acelaşi raport cu reducerea frecvenței de eşantionare, dată de inversul dimensiunii pre-imaginii pixelilor.

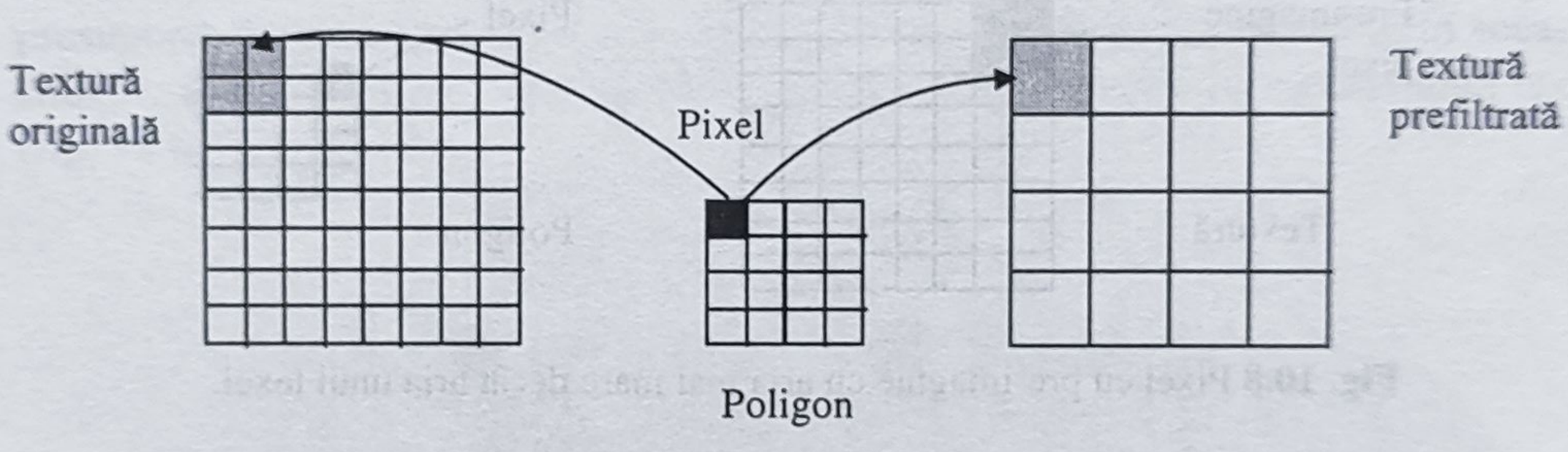


Fig. 10.9 Prefiltrarea texturilor.

O tehnică elegantă de prefiltrare a texturilor, care are o mare popularitate şi numeroase implementări hardware în sistemele grafice, este tehnica numită *mip-map*, dezvoltată de Wiliams în 1983. În loc să fie folosită o singură imagine de textură, în tehnica mip-map se foloseşte o succesiune de imagini ale aceleiaşi texturi, toate derivate din imaginea originală, prin medierea culorilor şi reducerea rezoluției. Fiecare imagine din succesiune are o rezoluție egală cu jumătatera rezoluției imaginii precedente şi fiecare texel este media a patru texeli din imaginea precedentă. Termenul *mip* provine din expresia *multum in parvo* (multe lucruri într-un spațiu restrâns). Parametrul L este parametrul de selecție a nivelului de filtrare a texturii. Nivelul L = 0 reprezintă imaginea originală a texturii (imaginea nefiltrată) cu rezoluția maximă. În general, în tehnica mip-map se folosesc rezoluții egale pe cele două coordonate s şi t, cu valoare o putere a lui 2, iar imaginile prefiltrate au rezoluții care scad prin înjumătățire până la valoarea 1×1 (fig. 10.10).

Prin selecția nivelului de filtrare a texturii (L), se alege imaginea cu rezoluția cea mai potrivită pentru dimensiunea pre-imaginii pixelului. Pentru evitarea discontinuităților între imagini la rezoluții diferite, se pot combina două

nivele de filtrare prin interpolare liniară. Selecţia corectă a nivelului L este importantă. Dacă L este prea mare, imaginea apare estompată; dacă L este prea mic, atunci se observă zgomotul de aliasing. Nivelul L se alege astfel încât aria pre-imaginii pixelului să fie cât mai apropiată de aria texelului.

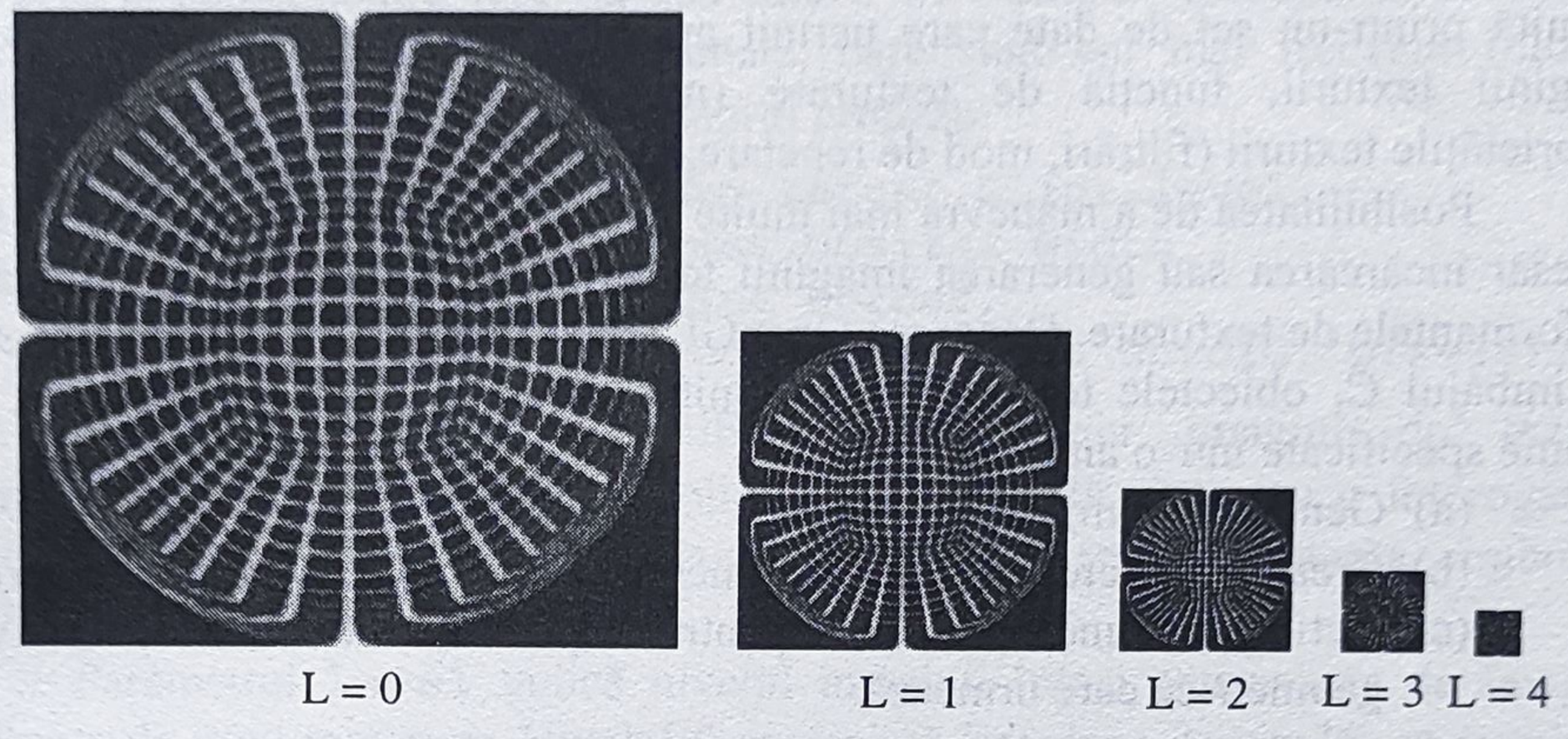


Fig. 10.10 Textură mip-map.

Detaliile de execuţie ale prefiltrării folosind texturi *mip-map* depind de implementarea specifică, hardware sau software. În subcapitolul următor sunt prezentate funcţiile bibliotecii OpenGL prin care se pot programa aplicaţii grafice folosind texturarea suprafeţelor.

10.3 FUNCŢII OPENGL DE TEXTURARE

Biblioteca OpenGL permite texturarea obiectelor, oferind suport pentru un număr foarte mare de posibilităţi de definire, aplicare şi filtrare a texturilor. În lucrare vor fi prezentate numai o parte dintre aceste posibilităţi, ca exemplificare a modului de implementare a texturării şi ca bază de pornire în crearea aplicaţiilor grafice. Celelalte aspecte, mai de detaliu, se pot urmări în manualul de referinţă al bibliotecii, precum şi în numeroasele exemple care sunt disponibile în OpenGL şi GLUT.

Se pot defini texturi unidimensionale sau bidimensionale, cu rezoluţii (număr de texeli) care se specifică la crearea imaginii texturii. Fiecare texel constă din una, două, trei sau patru componente, reprezentând valori de modulare sau cvadruple RGBA.

Texturarea poate fi validată sau invalidată prin apelul funcţiei `glEnable()` respectiv `glDisable()`, cu argument una din constantele simbolice `GLTEXTURE_1D` sau `GL_TEXTURE_2D` pentru texturi unidimensionale sau bidimensionale.

10.3.1 DEFINIREA TEXTURILOR

În OpenGL se pot defini mai multe texturi (în general, la inițializarea programului) și fiecare suprafață se texturează folosind una dintre acestea. Fiecare textură (obiect textură - *texture object*) are un nume unic în program și este definită printr-un set de date care permit aplicarea acesteia suprafețelor: adresa imaginii texturii, funcția de texturare (modulare, înlocuire, combinare), și proprietățile texturii (filtrări, mod de repetare, etc).

Posibilitatea de a manevra mai multe texturi în timpul execuției, fără să fie necesar încărcarea sau generarea imaginii texturii de fiecare dată, îmbunătățește performanțele de texturare. Deoarece OpenGL este o interfață de programare scrisă în limbajul C, obiectele textură sunt definite prin date și funcții separate, care trebuie specificate într-o anumită ordine:

(a) Generarea numelor texturilor.
(b) Crearea obiectelor textură și conectarea lor (*bind*) la numele texturilor.
(c) Activarea unei texturi, pentru aplicarea acesteia primitivelor geometrice care urmează.

Numele texturilor. Numele texturilor sunt numere întregi fără semn (de tipul `GLuint`) care sunt memorate într-un vector transmis ca argument funcției:

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

Această funcție creează un vector de nume de texturi, unice în program, pe care le memorează în vectorul `textureNames`. Numele create nu sunt neapărat numere succesive. Funcția `glGenTextures()` creează numai numele texturilor și le marchează ca utilizate, dar obiectele textură se creează numai la conectarea (*bind*) acestora.

Crearea texturilor. Funcția `glBindTexture()` se folosește atât pentru crearea, cât și pentru utilizarea unei texturi. Prototipul ei este:

```
void glBindTexture(GLenum target, GLuint texName);
```

Parametrul `target` poate avea ca valoare una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D` pentru texturi unidimensionale, respectiv bidimensionale. Argumentul `texName` este numele unei texturi, generat de funcția `glGenTextures()` și memorat în vectorul de nume ale texturilor. Atunci când este apelată prima oară pentru un nume de textură, funcția `glBindTexture()` creează un nou obiect textură, cu toate datele referitoare la imaginea și proprietățile texturii implicite. După apelul funcției `glBindTexture()`, textura cu numele dat ca argument devine textură curentă și toate operațiile ulterioare, atât pentru definirea unor proprietăți ale texturii, cât și pentru aplicația texturii, folosesc textura curentă. Textura curentă se schimbă prin apelul unei noi funcții `glBindTexture()`.

Crearea imaginii de textură. Imaginea texturii este un tablou unidimensional sau bidimensional de texeli, fiecare texel având una, două, trei sau

patru componente. Semnificaţia componentelor texelilor se stabileşte la crearea imaginii texturii prin definirea formatului intern al imaginii printr-un argument al uneia din funcţiile `glTexImage1D()`, `glTexImage2D()`.

Texturile unidimensionale au o utilizare restrânsă. Ele sunt folosite pentru texturarea în benzi, pentru care variaţia culorii are o singură direcţie. În continuare se vor prezenta texturile bidimensionale, care sunt cel mai frecvent folosite. Funcţia `glTexImage2D()` are următorul prototip:

```
void glTexImage2D(GLenum target,GLint level,
     GLint internalFormat,GLsizei width,
     GLsizei height,GLint border,
     GLenum format,GLenum type,const GLvoid *pixels);
```

Parametrul `target` specifică crearea unei imagini de textură bidimensională (prin constanta simbolică `GL_TEXTURE_2D`) sau o interogare a capacităţii de memorare a unei imagini de textură (prin constanta simbolică `GL_PROXY_TEXTURE_2D`).

Parametrul `level` indică nivelul imaginii într-o succesiune de imagini prefiltrate de tipul mip-map. Nivelul 0 reprezintă imaginea originală, cu rezoluţia maximă. Celelate nivele se pot crea automat din imaginea originală (prin funcţia `gluBuild2DMipmaps()`) sau pot fi create din tablouri de texeli, cu acelaşi format ca imaginea de nivel 0 a texturii.

Parametrul `internalFormat` indică tipul componentelor texelilor texturii, ca intensitate, luminanţă sau R, G, B, A. Valoarea argumentului `internalFormat` poate fi un întreg între 1 şi 4 sau una dintre treizeci şi două de constante simbolice, dintre care cele mai frecvent folosite sunt: `GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`, `GL_RGB`, `GL_RGBA`. Aceste constante corespund valorilor 1, 2, 3, 4 ale argumentului. Aceste componente ale texelilor sunt combinate cu componentele culorii proprii (înainte de texturare) a pixelului pentru a crea culoarea finală a pixelului. Modul de combinare depinde de funcţia de texturare, descrisă în continuare.

Parametrii `width` şi `height` dau dimensiunea imaginii texturii, în număr de texeli, pe orizontală şi pe verticală. Parametrul `border` reprezintă lăţimea borderului imaginii de textură. Această caracteristică permite alipirea mai multor porţiuni de imagini într-o singură imagine de textură, iar borderul este folosit în operaţiile de filtrare, în care sunt necesari texeli vecini texelului curent, pentru medierea culorilor acestora. Divizarea unei imagini de textură în mai multe părţi este impusă în anumite implementări ale bibliotecii OpenGL, atunci când texturarea este implementată hardware şi imaginea de textură se încarcă într-o memorie rapidă din sistemul grafic. Dacă imaginea texturii este prea mare pentru a fi stocată în întregime în memoria de textură, atunci se poate împărţi în mai multe părţi, care sunt încărcate şi folosite succesiv. În această situaţie borderul permite filtrarea corectă la graniţele părţilor componente ale imaginii.

Parametrii `format` şi `type` se referă la formatul şi tipul datelor imaginii de textură. Parametrul format poate fi una din constantele simbolice `GL_COLOR_INDEX`, `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`,

GL_ALPHA, GL_LUMINANCE, GL_LUMINANCE_ALPHA. Parametrul type poate fi GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_FLOAT, GL_INT, GL_UNSIGNED_INT, sau GL_BITMAP. Ultimul parametru, pixels, conține datele imaginii de textură.

La apelul funcției glTexImage2D() se creează imaginea texturii curente (cea conectată prin funcția glBindTexture()) în formatul specificat.

Funcții de texturare. Modul în care se calculează culoarea unui pixel (sau fragment de pixel) se stabileşte prin apelul uneia din funcțiile glTexEnv#():

```
void glTexEnvf(GLenum target,
               GLenum pname,GLfloat param);
void glTexEnvi(GLenum target,
               GLenum pname,GLfloat param);
```

unde argumentele au următoarea semnificație:

- target specifică mediul de aplicare a texturii; trebuie să fie constanta simbolică GL_TEXTURE_ENV
- pname este numele simbolic al unui paramentru de mediu de texturare; trebuie să aibă valoarea GL_TEXTURE_ENV_MODE
- param specifică funcția de texturare printr-o constantă simbolică care poate lua una din valorile GL_MODULATE, GL_DECAL, sau GL_BLEND.

Funcțiile de texturare stabilesc modul de calcul al culorii rezultante a pixelilor pe baza culorii texelilor din imaginea de textură şi a culorii pixelilor primitivei geometrice (care se obțin din culoarea curentă sau din calcule de umbrire). În general, în modul GL_DECAL culoarea finală atribuită unui pixel este culoarea texturii; în modul GL_MODULATE se foloseşte culoarea texturii pentru modularea culorii fragmentului (pixelului); în modul GL_BLEND se combină culoarea texturii cu culoarea fragmentului. Această modalitate generală de atribuire a culorii depinde şi de formatul intern al texturii, care se stabileşte la crearea imaginii de textură.

Atribuirea coordonatelor de texturare. Coordonatele vârfurilor primitivelor geometrice în planul texturii (coordonatele de texturare) se transmit prin funcțiile glTexCoord#(), care pot primi 1, 2, 3 sau 4 argumente de diferite tipuri, rezultând un număr mare de posibilități de apel. O parte dintre acestea sunt:

```
void glTexCoord1f(GLdouble s);
void glTexCoord2f(GLdouble s,GLdouble t);
void glTexCoord3f(GLdouble s,GLdouble t,GLdouble r);
void glTexCoord4f(GLdouble s,GLdouble t,GLdouble r,
                  GLdouble q);
void glTexCoord1f(GLfloat s);
void glTexCoord2f(GLfloat s,GLfloat t);
```

Coordonatele de texturare pot avea 1, 2 sau 3 valori pentru texturi unidimensionale, bidimensionale sau tridimensionale. În spaţiul texturii, coordonatele sunt notate `s`, `t`, `r`, corespunzător coordonatelor `x`, `y`, `z` în spaţiul obiect. Cea de-a patra componentă, `q`, este componenta de scală în reprezentarea texturii într-un sistem de coordonate omogen, asemănător sistemului de coordonate omogen folosit pentru reprezentarea punctelor în spaţiu. Această coordonată este folosită dacă sunt necesare transformări ale texturi în coordonate omogene. În versiunea folosită în momentul de faţă a bibliotecii OpenGL (Versiunea 1.1), nu este încă implementată texturarea spaţială şi coordonata `r` nu este folosită, fiind prevăzută ca rezervă pentru utilizare ulterioară.

Dacă valorile coordonatelor de texturare în vârfurile primitivelor grafice sunt cuprinse în intervalul [0,1], atunci textura este aplicată o singură dată pe suprafaţa respectivă. Dacă aceste valori depăşesc intervalul [0,1], atunci textura poate fi repetată pe suprafaţă sau limitată la intervalul [0,1]. Proprietatea unei texturi de a fi repetată sau limitată se stabileşte prin apelul uneia din funcţiile:

```
void glTexParameterf(GLenum target,GLenum pname,
                     GLfloat param);
void glTexParameteri(GLenum target,GLenum pname,
                     GLint param);
```

În aceste funcţii, parametrul `target` reprezintă tipul texturii şi poate lua una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D`. Parametrul `pname` specifică numele simbolic al unei proprietăţi a texturii. şi poate lua una din constantele: `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAX_FILTER`, `GL_TEXTURE_WRAP_S`, `GL_TEXTURE_WRAP_T`. Primele două valori se referă la opţiunile de filtrare ale texturii şi vor fi prezentate în paragraful următor. Următoarele valori setează proprietatea de repetare a texturii pentru coordonata `s`, respectiv `t`. În acest caz, parametrul `param` poate fi `GL_REPEAT` pentru repetarea texturii, sau `GL_CLAMP` pentru limitarea texturii la intervalul [0,1]. Mai multe aspecte privind definirea şi folosirea texturilor în OpenGL vor fi detaliate în exemplele care urmează.

■ Exemplul 10.1

În acest exemplu este prezentat programul prin care se aplică texturi în tablă de şah unor suprafeţe în spaţiu. Imaginea pe care o generează este o variantă a celei din fig. 10.2.

```
#include <GL/glut.h>

#define width 64
#define height 64

static GLubyte image4[height][width][4];
static GLubyte image8[height][width][4];
static GLuint texName[2];
```

```
void Makeimages(){
   int c;
   for (int i=0;i<height;i++){
      for (int j=0;j<width;j++){
         c = (((i&0x4)==0)^((j&0x4)==0)) *255;
         image4[i][j][0] = (GLubyte) c;
         image4[i][j][1] = (GLubyte) c;
         image4[i][j][2] = (GLubyte) c;
         image4[i][j][3] = (GLubyte) 255;
         c = (((i&0x8)==0)^((j&0x8)==0)) *255;
         image8[i][j][0] = (GLubyte) c;
         image8[i][j][1] = (GLubyte) c;
         image8[i][j][2] = (GLubyte) c;
         image8[i][j][3] = (GLubyte) 255;
      }
   }
}
void Init(void){
   glClearColor(0.6, 0.6, 0.6, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_FLAT);

   /* TEXTURARE */
   Makeimages();
   glPixelStorei(GL_UNPACK_ALIGNMENT,1);
   glGenTextures(2, texName); // se creaza doua texturi

   /* Definirea primei texturi */
   glBindTexture(GL_TEXTURE_2D,texName[0]);
   glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,width,
         height,0,GL_RGBA,GL_UNSIGNED_BYTE,image4);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
            GL_DECAL);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
                   GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
                   GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
                   GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                   GL_NEAREST);

   /* Definirea celui de-a doua texturi */
   glBindTexture(GL_TEXTURE_2D, texName[1]);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
      height, 0, GL_RGBA, GL_UNSIGNED_BYTE,image8);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
            GL_DECAL);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
                   GL_REPEAT);
```

```
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
                        GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
                        GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                        GL_NEAREST);
        glEnable(GL_TEXTURE_2D);
    }
    void Display(){
        glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glPushMatrix();
        glTranslated(-2.0,0.0,-8);
        glBegin(GL_QUADS);
           glTexCoord2f(0.0,0.0);glVertex3f(-0.5,-1.0,0.0);
           glTexCoord2f(0.5,0.0);glVertex3f( 0.5,-1.0,0.0);
           glTexCoord2f(0.5,1.0);glVertex3f( 0.5, 1.0,0.0);
           glTexCoord2f(0.0,1.0);glVertex3f(-0.5, 1.0,0.0);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslated(0.0,0.0,-8);
        glRotated(-70,1.0,0.0,0.0);
        glBegin(GL_QUADS);
           glTexCoord2f(0.0,0.0);glVertex3f(-0.5,-1.0,0.0);
           glTexCoord2f(0.5,0.0);glVertex3f( 0.5,-1.0,0.0);
           glTexCoord2f(0.5,1.0);glVertex3f( 0.5, 1.0,0.0);
           glTexCoord2f(0.0,1.0);glVertex3f(-0.5, 1.0,0.0);
        glEnd();
        glPopMatrix();
        glBindTexture(GL_TEXTURE_2D, texName[0]);
        glPushMatrix();
        glTranslated(1.5,0.0,-8);
        glBegin(GL_QUADS);
           glTexCoord2f(0.5,0.0);glVertex3f(0.0,-1.0,0.0);
           glTexCoord2f(1.0,0.0);glVertex3f( 1.0,-1.0,0.0);
           glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,0.0);
           glTexCoord2f(0.5,0.5);glVertex3f(-0.0, 0.0,0.0);
        glEnd();
        glPopMatrix();
        glutSwapBuffers();
    }
    void Reshape(int w, int h){
        h = (h == 0) ? 1 : h;
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(60.0, (GLfloat) w/(GLfloat) h,
                       0.1, 4000.0);
        glMatrixMode(GL_MODELVIEW);
```

```
        glLoadIdentity();
    }
    int main(int argc, char** argv){
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                            GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Texture Check Image");
        Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutMainLoop();
        return 0;
    }
```

Imaginea captată din fereastra afişată la execuţia acestui program este dată în fig. 10.11 şi diferă de imaginea din fig. 10.2 prin faptul că se folosesc două texturi.

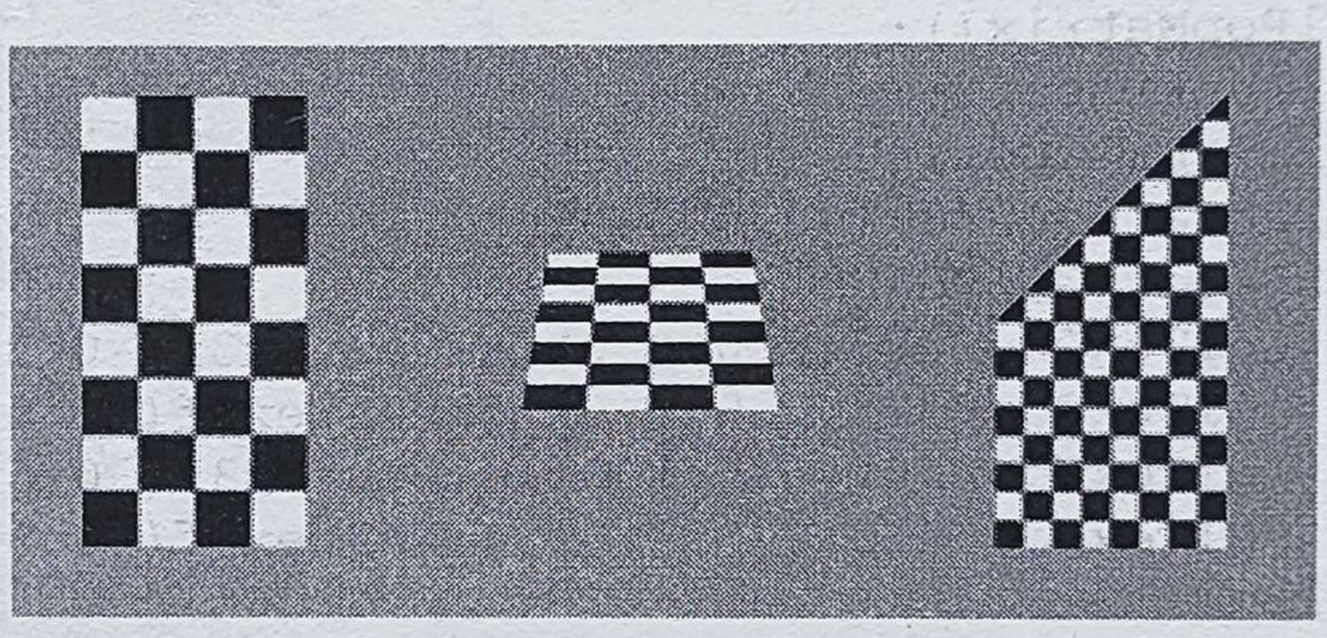


Fig. 10.11 Aplicaţia texturilor pe suprafeţe plane.

În mod obişnuit, imaginile texturilor se citesc din fişiere, dar texturile simple în tablă de şah se pot genera prin program. Funcţia `Makeimages()` creează două tablouri de 64 x 64 locaţii, fiecare locaţie fiind un vector cu patru componente R, G, B, A. În primul tablou (image4) este creată o tablă cu 16×16 pătrate alternante de culoare albă şi neagră, fiecare pătrat de dimensiune 4×4 texeli. În al doilea tablou (image8) este creată o tablă cu 8×8 pătrate alternante de culoare albă şi neagră, fiecare pătrat de dimensiune 8×8 texeli.

Texturile sunt definite în funcţia `Init()`. Mai întâi se creează numele a două texturi în vectorul `texName[2]`, prin funcţia `glGenTextures()`. Pentru crearea şi definirea proprietăţilor fiecărei texturi, se conectează mai întâi textura specificată prin numele ei (funcţia `glBindTexture()`) şi apoi se specifică proprietăţile texturii.

În funcţia `Display()` se utilizează texturile definite pentru texturarea unor primitive geometrice. Textura care se aplică este textura curentă, activată prin numele ei dat ca argument funcţiei de conectare `glBindTexture()`. Ca urmare,

primele două suprafeţe sunt texturate cu textura cu numele `texName[1]`, iar a treia suprafaţă este texturată cu textura cu numele `texName[0]`, ceea ce se poate observa în imaginea din fig. 10.11. Imaginea din fig. 10.2 a fost generată cu acelaşi program, dar s-a activat numai textura cu numele `texName[1]`, adică textura formată din 8 × 8 pătrate. Într-un bloc `glBegin()- .glEnd()` se transmit vârfurile unei primitive geometrice; pentru fiecare vârf se definesc mai întâi coordonatele de texturare (cu `glTexCoord2f()`) şi apoi coordonatele spaţiale ale vârfului (cu `glVertex3f()`). Restul funcţiilor din program efectuează operaţiile deja cunoscute, de transformări geometrice, setare culori de ştergere, etc.

10.3.2 STIVA MATRICELOR DE TEXTURARE

Cea dea-a treia stivă de matrice de transformare OpenGL este stiva matricelor de texturare, care se activează prin comanda `glMatrixMode(GL_TEXTURE)`. Matricea din vârful acestei stivei se aplică automat coordonatelor de texturare. În mod implicit, aceasta este matricea identitate şi coordonatele de texturare rămân nemodificate. Prin modificarea matricei din vârful stivei de texturare, se pot obţine de efecte de îngustare sau lărgire a texturii (cu `glScale#()`), de rotaţie (cu `glRotate#()`) sau translaţie (cu `glTranslate#()`) a texturii pe suprafaţa obiectelor. Dat fiind că matricea de texturare este o matrice 4 × 4, se pot introduce şi efecte de perspectivă în textură, dacă se folosesc coordonatele (s, t, r, q) de texturare.

■ **Exemplul 10.2**

În acest exemplu se evidenţiază folosirea stivei matricelor de texturare şi modul de repetare sau limitare a texturii. Programul este o versiune uşor modificată a programului din exemplul precedent. În continuare sunt redate numai funcţiile care s-au modificat: `Init()`, `Display()` şi `Reshape()`.

```
void Init(void){
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_CULL_FACE);
   glShadeModel(GL_FLAT);
   glClearColor(0.7, 0.7, 0.7, 1.0);
   Makeimages();
   glPixelStorei(GL_UNPACK_ALIGNMENT,1);
   glGenTextures(2, texName); // se creaza doua texturi
   /* Definirea primei texturi */
   glBindTexture(GL_TEXTURE_2D, texName[0]);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
       height, GL_RGBA, GL_UNSIGNED_BYTE,image4);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
             GL_DECAL);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                   GL_CLAMP);
```

```
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                        GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        /* Definirea celei de-a doua texturi */
        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
            height, 0, GL_RGBA, GL_UNSIGNED_BYTE,image8);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                  GL_DECAL);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                        GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                        GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glEnable(GL_TEXTURE_2D);
    }
    void Display(){
        glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
        glBindTexture(GL_TEXTURE_2D, texName[0]);
        glPushMatrix();
        glTranslated(-1.2,0.0,-8);
        glBegin(GL_QUADS);
           glTexCoord2f(0.0,0.0);glVertex3f(-1.0,-1.0,1.0);
           glTexCoord2f(1.0,0.0);glVertex3f( 1.0,-1.0,1.0);
           glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,1.0);
           glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,1.0);
        glEnd();
        glPopMatrix();
        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glPushMatrix();
        glTranslated(1.2,0,-8);
        glBegin(GL_QUADS);
           glTexCoord2f(0.0,0.0);glVertex3f(-1.0,-1.0,1.0);
           glTexCoord2f(1.0,0.0);glVertex3f( 1.0,-1.0,1.0);
           glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,1.0);
           glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,1.0);
        glEnd();
        glPopMatrix();
        glutSwapBuffers();
    }
    void Reshape(int w, int h){
        h = (h == 0) ? 1 : h;
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
    gluPerspective(60.0,(GLfloat)w/(GLfloat)h,0.1,4000);
        glMatrixMode(GL_TEXTURE);
        glLoadIdentity();
        glScaled(2,2,2);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
```

Imaginea generată la execuţia acestui program este prezentată în fig. 10.12.

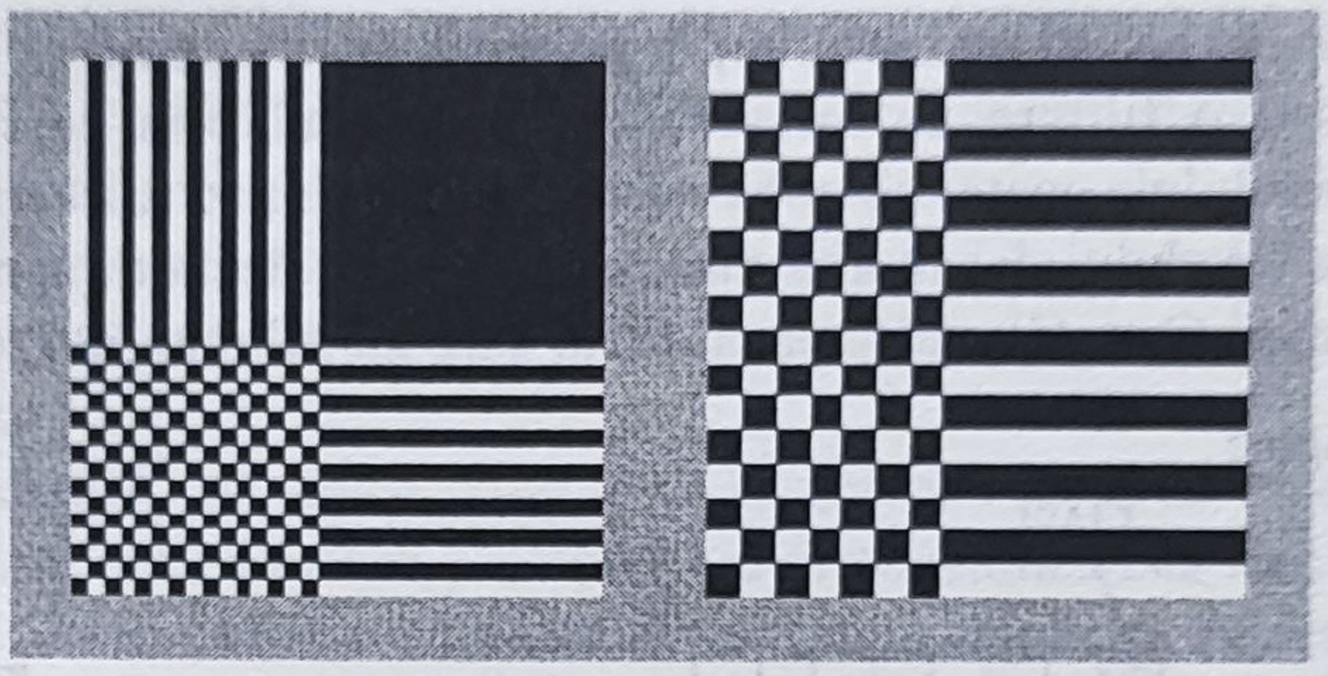


Fig. 10.12 Utilizarea stivei matricelor de texturare.
Repetarea şi limitarea texturilor.

În funcţia `Reshape()` se selectează stiva matricelor de texturare (`glMatrixMode(GL_TEXTURE)`) şi se introduce în vârful stivei o matrice de scalare (`glScaled(2.0,2.0,2.0)`). Această transformare se aplică coordonatelor de texturare ale celor două suprafeţe desenate şi deci aceste coordonate capătă valorile (0, 0), (0, 2), (2, 2), (2, 0). La depăşirea intervalului [0, 1], textura este repetată sau limitată, în funcţie de valoarea parametrilor `GL_TEXTURE_WRAP_S` şi `GL_TEXTURE_WRAP_T` ai texturii. Prima textură este limitată pe ambele coordonate `s` şi `t`, cea de-a doua textură este limitată pe coordonata `s` şi repetată pe coordonata `t`. Imaginea din figura de mai sus evidenţiază acest mod de aplicare a texturilor.

10.3.3 FILTRAREA TEXTURILOR

Tipul de filtrare care se aplică unei texturi este definit prin valorile a doi parametri, `GL_TEXTURE_MAG_FILTER` şi `GL_TEXTURE_MIN_FILTER`. setaţi prin apelul funcţiei `glTexParameter#()`. Filtrarea de mărire (*magnification*, `GL_TEXTURE_MAG_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este egală sau mai mică decât dimensiunea texelului (fig. 10.13(a)). Filtrarea de micşorare (*minification*, `GL_TEXTURE_MIN_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este mai mare decât dimensiunea texelului (fig. 10.13(b)).

Dacă nu este definită imagine mip-map a texturii, cei doi parametri de filtrare GL_TEXTURE_MAG_FILTER şi GL_TEXTURE_MAG_FILTER pot lua numai una din valorile GL_NEAREST sau GL_LINEAR. Valoarea GL_NEAREST înseamnă, de fapt, lipsa filtrării: se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului. Valoarea GL_LINEAR asigură filtrarea texturii prin calculul mediei ponderate a patru texeli cei mai apropiați de centrul pre-imaginii pixelului. Dacă este implementată software, filtrarea texturii (GL_LINEAR) este executată mai lent decât eşantionarea acesteia (GL_NEAREST).

În unele cazuri nu este simplu de decis dacă trebuie să se execute o filtrare de mărire sau o filtrare de micşorare. Patrulaterul prin care se reprezintă pre-imaginea pixelului poate să aibă dimensiunea într-o direcție mai mică decât dimensiunea texelului, iar în altă direcție mai mare decât dimensiunea texelului. În astfel de situații OpenGL selectează filtrul care dă cel mai bun rezultat posibil.

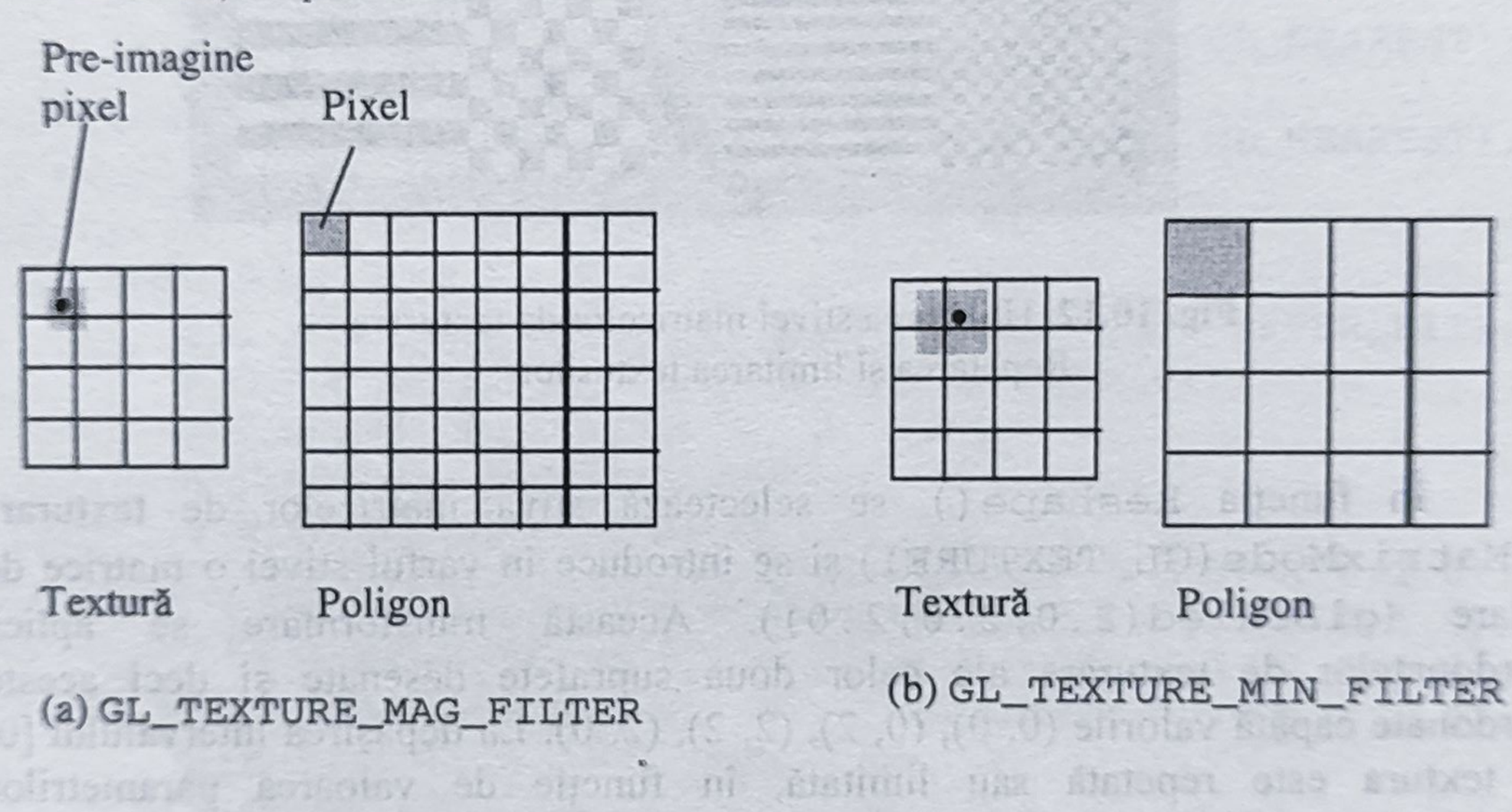


Fig. 10.13 (a) Filtrarea de mărire a texturii.
(b) Filtrarea de micşorare a texturii.

Filtrarea de micşorare este accelerată prin tehnica de prefiltrare a texturilor folosind imagini de texturi mip-map. Secvența de imagini ale texturii mip-map poate fi generată prin apelul funcției `glTexImage2D()` pentru fiecare nivel, începând cu nivelul 0 (rezoluție maximă) până la ultimul nivel, cu rezoluție 1×1. Datele fiecărei imagini (tabloul bidimensional de texeli) se obțin din imaginea precedentă prin înlocuirea fiecărui grup de patru texeli cu un texel a cărui culoare este media culorilor celor patru texeli din imaginea precedentă.

Dacă s-a definit imaginea mip-map a texturii, atunci filtrarea de micşorare se poate realiza în mai multe moduri, depinzând de felul în care se selectează nivelul imaginii mip-map şi de tipul de filtrare în imaginea mip-map selectată. Filtrul de micşorare (GL_TEXTURE_MIN_FILTER) este definit printr-una din următoarele constante simbolice:

- GL_NEAREST: nu se aplică nici-o filtrare, se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului din imaginea de nivel 0 a texturii mip-map;
- GL_LINEAR: se selectează imaginea de nivel 0 a texturii mip-map şi se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului;
- GL_NEAREST_MIPMAP_NEAREST: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului şi în această imagine se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului;
- GL_LINEAR_MIPMAP_NEAREST: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului şi în această imagine se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului;
- GL_NEAREST_MIPMAP_LINEAR: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul GL_NEAREST (texelul cel mai apropiat de centrul pre-imaginii pixelului); aceşti texeli se mediază ponderat pentru obţinerea valorii finale a culorii pixelului;
- GL_LINEAR_MIPMAP_LINEAR: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul GL_LINEAR (media ponderată a patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului) şi aceste valori se mediază ponderat pentru obţinerea valorii finale a culorii pixelului.

Filtrarea GL_LINEAR_MIPMAP_LINEAR, care se mai numeşte şi filtrare triliniară, este cel mai eficient mod de eliminare a zgomotului de aliasing al texturii, dar şi cel mai costisitor din punct de vedere al puterii de calcul. Chiar din simpla enumerare a operaţiilor efectuate, se poate observa cantitatea de calcule extrem de mare necesară pentru fiecare pixel al imaginii generate. Filtrarea triliniară a texturilor generează imagini deosebit de realiste, dar este fie lentă, dacă este executată software, fie costisitoare, dacă este implementată hardware.

În exemplul următor se prezintă modul de programare a filtrării texturilor şi rezultatele care se pot obţine cu diferite tipuri de filtrări.

■ **Exemplul 10.3**

Programul prin care s-au obţinut imaginile din Planşa 8 este următorul:

```
#include <gl/glaux.h>
#include <GL/glut.h>
```

```
AUX_RGBImageRec *image;
void Plane(){
   glBegin(GL_QUADS);
      glTexCoord2f(0.0,0.0);glVertex3f(-1.0, 1.0, 1.0);
      glTexCoord2f(1.0,0.0);glVertex3f( 1.0, 1.0, 1.0);
      glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,-1.0);
      glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,-1.0);
   glEnd();
}
void Init(void){
   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   image = auxRGBImageLoad("../textures/floor.rgb");
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
     image->sizeY,GL_RGB,GL_UNSIGNED_BYTE,image->data);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                GL_DECAL);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameterf(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_NEAREST);
                //GL_LINEAR_MIPMAP_LINEAR);
   glEnable(GL_TEXTURE_2D);
}
void Display(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
   glTranslatef(0.0, -2.2, -4.0);
   Plane();
   glPopMatrix();
   glutSwapBuffers();
}
void Reshape(int w, int h){
   h = (h == 0) ? 1 : h;
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60,(GLfloat)w/(GLfloat)h,0.1, 1000);
   glMatrixMode(GL_TEXTURE);
   glLoadIdentity();
   glScaled(8,8,8);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

```
int main(int argc, char** argv){
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                       GLUT_DEPTH);
   glutInitWindowSize(400,400);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Texturare");
   Init();
   glutDisplayFunc(Display);
   glutReshapeFunc(Reshape);
   glutMainLoop();
   return 0;
}
```

În funcţia `Reshape()` se încarcă matricea din vârful stivei matricelor de texturare cu o matrice de scalare care multiplică coordonatele de texturare cu factorul 8. În acest program se foloseşte o singură textură, care este în permanenţă textura curentă. Toate funcţiile de definire a parametrilor şi de aplicare a texturii se referă la acestă unică textură şi nu a mai fost necesar atribuirea unui nume şi conectarea (*bind*) texturii pentru definire şi activare.

Imaginea texturii se citeşte dintr-un fişier, folosind funcţia `auxRGBImageLoad()` care aparţine unei biblioteci auxiliare din OpenGL, `glaux.lib`, care nu a fost încă descrisă. Acestă bibliotecă auxiliară este o versiune mai veche a sistemului de dezvoltare `GLUT`, care a înlocuit-o pentru cea mai mare parte din funcţiile de creare a ferestrei de afişare OpenGL, tratare a evenimentelor din sistem sau generare a unor obiecte tridimensionale. Deoarece în versiunea 3.6 a bibliotecii GLUT în care au fost scrise programele, nu există o funcţie de încărcare a imaginilor de textură, s-a apelat funcţia din glaux.lib. Prin funcţii ale bibliotecii glaux.lib se pot încărca imagini de textură în format RGB sau în format bitmap DIB (*Device Independent Bitmap*).

Imaginea citită din fişier este folosită pentru crearea secvenţei mip-map de imagini prefiltrate ale texturii. Acestă operaţie se poate efectua prin funcţia `gluBuild2DMipmaps()`.

După crearea imaginii mip-map a texturii, operaţiile de definire şi aplicaţie a texturii sunt asemănătoare cu cele prezentate în exemplele precedente. Setarea tipului de filtrare se face în funcţia `Init()` prin funcţia `glTexParameterf()`.

Pentru imaginea din Planşa 8 (a) nu se aplică nici o filtrare de micşorare prin apelul funcţiei:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_NEAREST);
```

Pentru imaginea din Planşa 8 (b) se aplică modul de filtrare triliniară prin apelul funcţiei:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
```

În Planşa 8 se observă diferența dintre imaginea cu textură filtrată şi imaginea cu textură nefiltrată.

10.3.4 GENERAREA COORDONATELOR DE TEXTURARE

În general, coordonatele de texturare se generează la crearea modelelor obiectelor tridimensionale. Pentru obiectele modelate prin fețe poligonale, fiecare vârf se defineşte prin coordonate, normală şi coordonate de texturare. Aceste date sunt transmise ca date ale primitivelor geometrice (în blocul `glBegin()` – `glEnd()`) şi prelucrate corespunzător.

OpenGL este o bibliotecă de redare, nu de modelare a obiectelor tridimensionale, ea nu are funcții de generare şi stocare a modelelor obiectelor care să conțină, printre alte informații, şi coordonatele de texturare ale obiectelor. Pentru crearea modelelor se folosesc sisteme de dezvoltare specializate, iar comenzile de generare a coordonatelor de texturare din OpenGL sunt utile numai în anumite situații, de exemplu, pentru generarea texturilor în mişcare, care necesită coordonate de texturare calculate într-un plan a cărui poziție depinde de punctul de observare.

Funcțiile de generare a coordonatelor de texturare sunt `glTexGen#()` cu mai multe variante în funcție de tipul argumentelor:

```
void glTexGend(GLenum coord, GLenum pname,
               GLdouble param);
void glTexGenf(GLenum coord, GLenum pname,
               GLfloat param);
void glTexGeni(GLenum coord,GLenum pname,GLint param);
```

Primul parametru, `coord`, specifică coordonata de texturare care se generează şi poate lua una din valorile constante: `GL_S`, `GL_T`, `GL_R` sau `GL_Q`, pentru coordonatele s, t, r, q respectiv. Parametrul `pname` este numele simbolic (`GL_TEXTURE_GEN_MODE`) al funcției de generare a coordonatelor de texturare, iar parametrul `param` poate lua una din valorile simbolice `GL_OBJECT_LINEAR`, `GL_EYE_LINEAR`, `GL_SPHERE_MAP`.

Pentru parametrii `GL_OBJECT_LINEAR` şi `GL_EYE_LINEAR` funcția de generare a coordonatelor de texturare realizează o aplicație a texturii în două etape folosind ca suprafață intermediară un plan. În modul `GL_OBJECT_LINEAR` planul este specificat în sistemul de referință de modelare şi se obține o textură fixă pe obiect. În modul `GL_EYE_LINEAR` planul intermediar este specificat în sistemul de referință de observare şi se poate obține o textură dinamică, care se mişcă pe obiect.

Parametrul `GL_SPHERE_MAP` defineşte o funcție de generare a coordonatelor de texturare printr-o aplicație în două etape folosind ca suprafață intermediară o suprafață sferică. Aplicația sferică a texturii dă posibilitatea de a reprezenta obiecte din materiale care simulează materiale reale (de exemplu, lemn, marmură, rocă, etc).

Aplicaţia unei coordonate de texturare generate de OpenGL este efectivă numai dacă a fost validată prin apelul uneia dintre funcţiile `glEnable(GL_TEXTURE_GEN_S)` sau `glEnable(GL_TEXTURE_GEN_T)` pentru coordonata `s`, respectiv pentru coordonata `t`.

■ **Exemplul 10.4**

În programul care urmează este exemplificată aplicaţia sferică a unei texturi bidimensionale.

```
#include <gl\glaux.h>
#include <gl\glut.h>
AUX_RGBImageRec *image;
void Cube(){
    // aceeasi funcţie ca în exemplul precedent,
    // cu deosebirea că laturile sunt egale
}
void Init(){
     glClearColor(0.6,0.6,0.6,1.0);
     image = auxRGBImageLoad("../textures/circles.rgb");

     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
     gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
      image->sizeY,GL_RGB,GL_UNSIGNED_BYTE,image->data);
     glEnable(GL_TEXTURE_2D);
     glEnable(GL_DEPTH_TEST);
     glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
                GL_SPHERE_MAP);
     glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
                GL_SPHERE_MAP);
     glTexParameterf(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
     glTexParameterf(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                GL_REPEAT);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                GL_REPEAT);
     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                GL_DECAL);
}
void  Display(){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glPushMatrix();
    glTranslatef(-2.0, -2.5, -8);
    glRotatef(30,1,0,0);
```

```
    glRotatef(20,0,1,0);
    glutSolidTeapot(1.4);
    glPopMatrix();
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);
    glPushMatrix();
    glTranslatef(2.0, -2.5, -8);
    glRotatef(30,1,0,0);
    glRotatef(20,0,1,0);
    Cube();
    glPopMatrix();
    glutSwapBuffers();
}
```

Funcțiile `main()` şi `Reshape()` ale programului sunt aceleaşi cu cele din exemplul 10.2 şi nu au mai fost prezentate din nou. Imaginea care se obține la execuția acestui program este dată în Planşa 9(a).

În acest program se foloseşte o singură textură mip-map cu filtrare de mărire (`GL_LINEAR`) şi de micşorare (`GL_LINEAR_MIPMAP_LINEAR`). Pentru ceainic (Planşa 9(a)) se validează generarea automată a coordonatelor de texturare cu aplicație sferică (`GL_SPHERE_MAP`). Pentru cub (Planşa 9(b)) se transmit coordonatele de texturare prin comenzile `glTexCoord2f()` şi este invalidată generarea automată a coordonatelor de texturare (`glDisable(GL_TEXTURE_GEN_S)`, `glDisable(GL_TEXTURE_GEN_T)`).

Texturarea prin aplicație sferică creează efecte deosebit de interesante. Ea este folosită frecvent în ceea ce se numeşte *aplicație a mediului* (*environment mapping*) prin care se redă un obiect perfect reflectiv, în care se reflectă toate obiectele din mediul înconjurător.

11

APLICAȚII ALE REALITĂȚII VIRTUALE

Orice aplicație de realitate virtuală se desfăşoară într-o scenă virtuală compusă dintr-o colecție de modele de obiecte tridimensionale. În mod obişnuit, crearea scenelor virtuale (*modelarea*) şi vizualizarea (*redarea*) acestora se desfăşoară ca două procese separate, datorită cerințelor de timp de execuție foarte diferențiate.

Modelarea scenelor virtuale este un proces off-line prin care se creează colecția de modele ale obiectelor care compun scena virtuală. Procesul de modelare a obiectelor se desfăşoară asistat de calculator, în diferite sisteme de proiectare interactivă, care pun la dispoziția proiectantului funcții de creare a modelelor pornind de la diferite date de intrare.

Pentru aplicațiile de realitate virtuală pot fi folosite atât sisteme de proiectare automată de uz general, cum sunt, de exemplu Autocad sau 3DStudio, sau sisteme de proiectare special concepute pentru crearea scenelor virtuale, numite sisteme de generare a bazelor de date (DBGS – *Data Base Generation Systems*). Astfel de sisteme, cum sunt sistemul Multigen, Corypheus, Sense8 şi multe altele, oferă numeroase posibilități de modelare a obiectelor tridimensionale, a terenului, şi a fenomenelor atmosferice (nori, ceață, etc.). Rezultatul operației de modelare îl reprezintă modelele obiectelor sau colecțiile de modele care compun o scenă virtuală, şi acestea sunt memorate sub forma de *baze de date grafice*, alcătuite dintr-unul sau mai multe fişiere.

Redarea scenelor virtuale este un proces care se desfăşoară interactiv, de cele mai multe ori în timp real, în cursul diferitelor experimente de realitate virtuală (simulatoare de antrenament, muzee virtuale, etc.), în sisteme grafice numite generatoare de imagine (*Image Generators* –IG). În generatoarele de imagine se re-creează modelele obiectelor şi al scenei virtuale, folosind ca date de intrare fişierele bazei de date grafice. Pentru redarea eficientă a imaginii, scena virtuală este organizată ierarhic, pe mai multe nivele de ierarhie, ceea ce permite selectarea rapidă a obiectelor, moştenirea transformărilor geometrice şi a atributelor, etc. Această organizare ierarhică se implementează printr-un graf aciclic direcționat, numit *graful scenei* (*scene graph*). Nodurile grafului conțin modele ale obiectelor sau informații de redare (poziția obiectelor, materiale, lumini, etc.).

11.1 BAZE DE DATE GRAFICE

Bazele de date grafice sunt reprezentarea pe suport de memorie (fişiere) a modelelor obiectelor şi a scenelor virtuale şi conţin toate informaţiile necesare pentru redarea scenelor.

Bazele de date obişnuite, utilizate pentru stocarea, actualizarea şi regăsirea unor informaţii după un anumit criteriu, se bazează pe diferite modele de date, ca de exemplu modelul ierarhic, modelul reţea, modelul relaţional, modelul orientării pe obiecte. În momentul de faţă, majoritatea sistemelor de baze de date sunt sisteme relaţionale care asigură eficienţă şi fiabilitate în actualizarea şi regăsirea informaţiilor memorate.

În cazul bazelor de date grafice, criteriul principal de selectare a unui model de date este acela de a asigura redarea eficientă a scenei virtuale în cadrul aplicaţiilor de timp real. Datorită reprezentării sub formă de graf a scenei virtuale în majoritatea generatoarelor de imagine, bazele de date grafice sunt, în majoritatea cazurilor, organizate într-un model ierarhic care foloseşte reprezentarea relaţiilor între entităţi prin legături funcţionale explicite de tipul părinte-fiu. Modelul ierarhic al bazelor de date grafice este cel mai adecvat modului de utilizare a datelor pe care acestea le conţin.

Există un număr mare de formate de reprezentare a bazelor de date grafice, mai mult sau mai puţin complete din punct de vedere al informaţiilor de construire a grafului scenei virtuale. Proliferarea formatelor bazelor de date îngreunează portabilitatea programelor de aplicaţii grafice, care trebuie să prevadă posibilităţi de interpretare pentru cât mai multe formate.

În orice generator de imagine, modelul scenei are o structură bine precizată, care depinde de scopul aplicaţiei, iar datele importate din diferite formate de baze de date sunt convertite la formatul intern al scenei. Un tip de format de bază de date reprezintă un anumit mod de descriere a entităţilor componente ale bazei de date şi a relaţiilor dintre ele. Cele mai cunoscute dintre formatele de baze de date grafice sunt prezentate în tabelul 11.1.

Tabelul 11.1

Formate de baze de date grafice

Denumire	Descriere
3ds	Format binar 3DStudio, firma AutoDesk
dxf	Format ASCII AutoCad, firma AutoDesk
obj	Format ASCII, firma Wavefront Technologies
phd	Format de descriere poliedre, firma Silicon Graphics
iv	Format ASCII Open Inventor, firma Silicon Graphics
dwb	Format binar/ASCII, firma Corypheus
flt14	Format binar Open Flight, firma Multigen

Formatele *3ds* şi *dxf* sunt formate de export ale sistemelor de proiectare de uz general 3DStudio şi AutoCad de la firma AutoDesk. Ele nu au fost concepute special pentru proiectarea bazelor de date grafice, astfel că nu pot avea decât o utilizare limitată. De exemplu, se pot folosi pentru proiectarea unor obiecte tridimensionale individuale, care sunt apoi înglobate în scene descrise prin formate mai complete.

Formatul *obj* este un format simplu care poate descrie obiecte tridimensionale reprezentate prin rețea de poligoane. Firma Wavefront Technology a plasat în domeniu public un număr de modele în format *obj*, care a devenit în felul acesta destul de cunoscut. O bogată librărie de modele produse de firma Viewpoint (peste 3500 de modele tridimensionale reprezentând avioane, maşini, animale, organe anatomice, etc.) sunt disponibile în formate *dxf*, *obj*, *flt*. Modelul avionului F-16 din fig. 2.16 este un model în formatul *obj*.

Formatele *dwb* şi *flt* sunt formate speciale de descriere a bazelor de date grafice, dezvoltate împreună cu sistemele de generare a bazelor de date de companiile Corypheus şi, respectiv, Multigen. Aceste formate conțin informații complete privind obiectele, amplasarea şi gruparea lor, materiale, lumini, din care se poate crea graful scenei în generatorul de imagine.

Formatul *iv* (Open Inventor), dezvoltat de firma Silicon Graphics, este un superset al limbajului de modelare VRML 1.0 (*Virtual Reality Moddeling Language*), care în 1997 a devenit standard sub numele de VRML 97. Limbajul VRML 97 permite crearea scenelor virtuale şi accesarea lor prin internet. O scenă virtuală în limbajul VRML este un format de bază de date grafice, asemănător altor formate de baze de date. Denumirea de limbaj VRML provine din faptul că specificațiile VRML conțin reguli sintactice şi semantice precise, care permit verificarea corectitudinii construcțiilor folosite. Acest limbaj este descris în subcapitolul următor.

11.2 CREAREA ŞI REDAREA SCENELOR VIRTUALE

O scenă virtuală (*virtual scene*) este compusă dintr-o colecție de modele de obiecte tridimensionale, specificate prin forma şi poziția lor, prin aspect (culoare, material, etc), şi comportare (deplasare în spațiu, interacțiune). Diferite alte denumiri mai sunt utilizate cu acelaşi înțeles: lume virtuală *(virtual world)*, mediu virtual (*virtual environment*). Construirea ierarhică a scenei permite organizarea şi parcurgerea eficientă a acesteia. Scena ierarhică este compusă din noduri conectate prin arcuri direcționate. Un nod în scenă descrie o anumită entitate: un obiect tridimensional, o grupare de obiecte tridimensionale, o transformare geometrică, o textură, etc. Un arc al grafului scenei introduce descendenți (fii) ai unui nod, care moştenesc unele din atributele nodului părinte (fig. 11.1).

Construirea ierarhică a scenei permite reutilizarea obiectelor şi a transformărilor geometrice. Un obiect complex se compune din gruparea mai

multor obiecte mai simple, dintre care unele se repetă, în poziţii de instanţiere diferite. În felul acesta, un nod poate avea mai mult de un nod părinte.

Fig. 11.1 Graful unui scene virtuale reprezentat printr-o ierarhie de noduri.

Nodurile grafului sunt de mai multe tipuri, dintre care cele mai importante sunt: nodurile de descriere a formei geometrice a obiectelor tridimensionale, nodurile de transformare şi nodurile de grupare. Nodurile de descriere geometrică conţin modele ale unor obiecte elementare, reprezentate poligonal, parametric sau prin ecauţia matematică într-un sistem de referinţă local (de modelare). Nodurile de transformare poziţionează un nod fiu faţă de nodul părinte. Nodurile de grupare formează obiecte compuse din mai multe noduri fii.

În fig.11.1 nodurile A, B, C, D, E, F reprezintă obiecte tridimensionale. Nodul G_1 grupează mai multe obiecte simple, formând un obiect compus care, la rândul lui, este instanţiat de două ori, în poziţii definite prin intermediul nodurilor de transformare T_1 şi T_2, rezultând obiecul O_1, s.a.m.d. Nodul R este un nod de grupare şi reprezintă rădăcina grafului scenei.

Graful scenei descrie relaţiile ierarhice între noduri. Atributele unui obiect descris de un nod pot fi cele moştenite de la nodul părinte, pot fi modificate, sau redefinite în nodul fiu. Scena virtuală este definită în sistemul de referinţă universal, în care se poziţionează şi orientează punctul de observare în fiecare cadru al imaginii generate. Obiectele din nodurile frunză sunt reprezentate într-un sistem de referinţă local. Un nod de transformare conţine o matrice de transformare **M** care defineşte poziţia şi orientarea sistemului de referinţă al nodului fiu în sistemul de referinţă al nodului părinte. Transformarea prin aplicarea matricei **M** punctelor obiectelor din nodul fiu le transformă din sistemul de referinţă al nodului fiu în sistemul de referinţă al nodului părinte.

Redarea imaginii unei scene virtuale reprezentate printr-un graf aciclic direcționat se realizează prin parcurgerea (*traversarea*) grafului scenei. Traversarea grafului scenei este o operație de parcurgere a grafului în adâncime (*depth first search*), care începe cu nodul rădăcină al grafului şi parcurge toate drumurile în graf până la fiecare nod frunză. În acest fel, un nod poate fi traversat de mai multe ori, pe căi diferite de parcurgere. De exemplu, la parcurgerea grafului din fig. 11.1, nodul A este traversat de două ori, o dată pe calea $RT_1O_1T_{11}G_1A$ şi a doua oară pe calea $RT_1O_1T_{12}G_1A$; la fel şi nodurile B şi C.

În reprezentarea ierarhică a obiectelor şi a scenei, un obiect tridimensional este instanțiat (plasat în sistemul de referință universal) prin aplicarea unei secvențe de transformări de instanțiere, şi anume toate transformările din nodurile de transformare de la rădăcină până la nodul frunză pe una din căile de parcurgere a grafului. Combinarea dintre operația de parcurgere în adâncime a grafului şi operațiile de concatenare a matricelor de transformare prin intermediul stivelor de transformare permite reutilizarea matricelor calculate pe parcursul traversării prin salvarea lor în stivele de transformări şi restaurarea atunci când sunt necesare.

Algoritmul de traversare a grafului scenei prezentat în continuare (în pseudocod asemănător limbajului C) specifică parcurgerea în adâncime a grafului scenei şi operațiile care se execută la vizitarea fiecărui nod, în funcție de tipul acestuia. Se consideră un tip de date `Node`, care poate fi o clasă de obiecte (în programarea orientată pe obiecte) sau o structură (în abordarea procedurală) şi o stivă de matrice de transformare de modelare, care admite operațiile de introducere (`glPushMatrix()`), de extragere (`glPopMatrix()`) şi de concatenare (`glMultMatrix()`) a matricei curente de transformare **C** cu o nouă matrice **M**, cu depunerea rezultatului în matricea curentă (**C** = **C M**). În mod obişnuit, aceasta este stiva OpenGL de modelare-vizualizare (stiva `MODELVIEW`). În această prezentare se consideră trei tipuri de noduri: noduri frunză (`SHAPE`), noduri grup (`GROUP`) şi noduri de transformare (`TRANSFORM`).

```
void Traversal (Node* node){
   if (node->type == SHAPE){
      Redare obiect folosind stiva curenta C
   }
   else if (node->type == GROUP){
      for (i=0; i<nChilds; i++){
         glPushMatrix();
            Traversal(node->child[i]);
         glPopMatrix();
      }
   }
   else if (node->type == TRANSFORM){
      glMultMatrix(M);
      glPushMatrix();
         Traversal(node->child[i]);
      glPopMatrix();
   }
}
```

Notaţiile care s-au folosit în acest algoritm sunt evidente. Tipul nodului (`node->type`) determină modul de prelucrare al acestuia. Un nod de grupare are `nChilds` noduri fii, numerotaţi `childs[i]`. Un nod de transformare conţine o matrice de transformare **M**. Travesarea începe prin iniţializarea matricei curente de transformare cu matricea identitate (**C** = **I**) şi apelul funcţiei de traversare pentru nodul rădăcină al scenei (`Traversal(R)`).

Matricele de instanţiere se introduc în stivă în ordinea inversă aplicării lor asupra obiectului, aşa cum este necesar în operaţia de acumulare prin postmultiplicare. De exemplu, pentru prima instanţă a obiectului A, se execută operaţiile:

$$\mathbf{C} = \mathbf{I}$$
$$\mathbf{C} = \mathbf{C}\,\mathbf{M}_1 = \mathbf{M}_1$$
$$\mathbf{C} = \mathbf{C}\,\mathbf{M}_{12} = \mathbf{M}_1\mathbf{M}_{12}$$

Un punct oarecare P al obiectului A este instanţiat prin transformarea:

$$\mathbf{P'} = \mathbf{CP} = \mathbf{M}_1\mathbf{M}_{12}\mathbf{P}$$

Pentru o altă instanţiere a nodului A, matricea intermediară $\mathbf{M}_1$ este salvată înainte de prelucrarea fiecăruia dintre fiii nodului O_1. Poate este mai puţin evidentă avantajul metodei în acest exemplu foarte mic, dar pentru scene complexe, cu multe transformări succesive, economia de calcule de produse de matrice este considerabilă.

Acesta este modul principial de organizare şi traversare a grafului unei scene virtuale. Construirea unui astfel de graf se poate face fie direct, prin program (pentru mici exemple cu scene de dimensiuni reduse) sau, în forma cea mai generală, prin parcurgerea unei baze de date descrise într-un limbaj (format) care să permită identificarea construcţiei ierarhice a scenei: tipuri de noduri, atributele acestora, relaţiile ierarhice între noduri, etc.

11.2.1 CREAREA GRAFULUI SCENEI

În implementările actuale se utilizează frecvent programarea orientată pe obiecte pentru reprezentarea scenei virtuale, abordare care permite crearea unui program eficient, uşor de exploatat şi de întreţinut.

În continuare este prezentată o astfel de metodă de construire a grafului scenei, ca un proiect simplu şi uşor de urmărit. Soluţia adoptată pentru acest proiect (care va fi numit CSV – *Crearea Scenelor Virtuale*) urmăreşte modul de definire şi creare a grafului scenei folosit în limbajul VRML şi în mai multe din sistemele de creare a scenelor virtuale disponibile în momentul de faţă. Exemple de astfel se sisteme sunt: *Performer* (de la firma Silicon Graphics), *WorldUp* (de la firma Sense8), *dVS* (de la firma Division) *EasyScene* (de la firma Corypheus), *Multigen* (de la firma Software Systems, numită ulterior Multigen). Denumirile date claselor de obiecte în proiectul CSV sunt asemănătoare celor folosite în astfel de sisteme (`Node`, `Shape`, etc.). S-a considerat că folosirea unor denumiri în limba română ar

îngreuna remarcarea similitudinor obiectelor proiectului cu cele din sistemele cele mai folosite de creare şi redare a scenelor virtuale.

Pentru descrierea orientată pe obiecte a grafului scenei virtuale se creează o ierarhie de clase, toate derivate dintr-o clasă de bază unică, numită clasa `Object`. Folosirea unei clase de bază unice pentru toate clasele permite definirea unui număr redus de clase pentru descriere diferitelor tipuri de liste liniare necesare pentru reprezentarea listelor de coordonate, de fețe, etc.

11.2.1.1 Vectorii de date

Clasa `ObArray` conține un vector de pointeri la clasa `Object`, alocat dinamic în memoria liberă. Reprezentarea prin vector a listelor de entități ale modelului (liste de coordonate de vârfuri, liste de fețe, etc.) este cea mai eficientă reprezentare pentru operația de prelucrare a listelor liniare, datorită accesului direct la elementele vectorului. Vectorul definit de clasa `ObArray` este vector cu auto-creştere dinamică, adică se realocă cu o dimensiune mai mare dacă se depăşeşte dimensiunea alocată la un moment dat. Chiar dacă operația de realocare consumă timp de execuție ridicat, acest lucru nu este supărător deoarece vectorii se creează o singură dată, la crearea grafului scenei, înainte de experiența interactivă în realitatea virtuală.

```
class Object{
public:
   Object() {};
   ~Object() {};
};
class ObArray : public Object {
   int size;          // dimensiune curenta
   int grows;         // interval de crestere a dimens.
   int dimens;        // dimensiune alocata
   Object **p;        // vector de pointeri la Object
public:
   ObArray(){         // constructor implicit
      size = 0;
      grows = 4;
      dimens = grows;
      p = (Object **)new BYTE[grows * sizeof(Object*)];
   }
   ~ObArray();
   int Add(Object* x);
   int InsertAt(int i, Object *x);
   int RemoveAt(int i);
   ..........................................
};
```

În mod asemănător se defineşte clasa `IntArray`, pentru vectori de numere întregi şi clasa `DoubleArray`, pentru vectori de numere în virgulă flotantă dublă precizie.

```
class IntArray {
    int size;
    int grows;
    int dimens;
    int *p;
public:
    IntArray() {
        size = 0;
        grows = 4;
        dimens = grows;
        p = new int[grows];
    }
    ~IntArray() { delete p;}
    int GetSize() {return size;}
    int Add(int x);
    int GetAt(int i){return p[i];}
    int RemoveAt(int i);
    int& operator[](int i){return p[i];}
};
```

Definiţia clasei `DoubleArray` este similară clasei `IntArray`.

Tot ca tipuri de date necesare pentru reprezentarea grafului se mai definesc clase pentru vectori de coordonate, cu două, trei, şi patru dimensiuni: clasele `Vect2`, `Vect3`, `Vect4`, toate derivate dintr-o clasă de bază `Vect`, ale căror date şi funcţii membre mai importante sunt date mai jos:

```
class Vect : public Object{
    double *vect;          // vectorul de date
    int d;                 // dimensiune vector
public:
    Vect(int s, int *p); // constructor
    virtual ~Vect(){delete vect;}
    double& operator[](int i){return vect[i];}
    Vect operator=(Vect v);
    Vect operator+=(Vect v);
    double *Get() const {return vect;}
    double Get(int i)const {return vect[i];}
    void Set(int i, double v){vect[i]=v;}
    void Set(double *v);
};
Vect::Vect(int s, double *p){
    d = s;
    vect = new double [d];
    for (int i=0; i<d; i++)
        vect[i] = p[i];
}
class Vect2 : public Vect {
public:
Vect2(double *p): Vect(2,p){ }
Vect2 operator=(Vect2 v);
```

```
    Vect2 operator+=(Vect2 v);
};
class Vect3 : public Vect {
public:
    Vect3(double *p) : Vect(3, p){ }
    Vect3 operator=(Vect3 v);
    Vect3 operator+=(Vect3 v);
};
```

Aceste clase definite în proiect pot fi înlocuite cu clase similare ale bibliotecii de clase a compilatorului folosit sau cu clase template. De exemplu, dacă se foloseşte compilatorul Microsoft Visual C++ (versiuni 4.2, 5.0, 6.0), atunci se pot prelua clase din biblioteca de clase MFC 3.0 (*Microsoft Foundation Class*), care implementează tipurile de colecţii de date. Clasa `Object` corespunde (cu multe simplificări) clasei `CObject` din MFC, clasa `ObArray` corespunde clasei `CObArray`, clasa `IntArray` corespunde clasei `CUIntArray`.

11.2.1.2 Nodurile grafului

Nodurile grafului scenei sunt reprezentate printr-o ierarhie de clase de obiecte care modelează comportarea acestora şi relaţiile de moştenire dintre ele. Clasa de bază pentru toate nodurile grafului este clasa `Node`, care este o clasă abstractă, iar toate celelelte clase care descriu nodurile grafului sunt derivate din aceasta.

```
class Node : public Object{
    ObArray parents;
    Vect3 bmin;    // volumul de delimitare
    Vect3 bmax;    // (bounding box)
public:
    Node(){}
    ~Node();
    int Invisible();
    virtual void Draw()=0;
};
```

Clasa `Node` descrie comportarea generală a unui nod în graf prin definirea unui vector de pointeri la nodurile părinte (`parents`) şi o funcţie virtuală pură de redare (funcţia `Draw()`), care se va redefini în fiecare clasă concretă derivată din clasa `Node`. Numărul de părinţi ai unui nod este folosit în operaţiile de ştergere a nodurilor grafului: un nod eliminat dintr-o poziţie a grafului nu poate fi şters din memorie decât dacă numărul de părinţi este egal cu zero.

Tot în clasa `Node` este implementat mecanismul de eliminare a obiectelor la redare prin testarea volumului de delimitare. Volumul de delimitare este definit în acest proiect ca un paralelipiped dreptunghic dat prin coordonatele minime şi maxime pe cele trei axe de coordonate (grupate în vectorii `bmin` şi `bmax`).

În cursul redării imaginii, dacă volumul de delimitare este sigur invizibil (se află în afara volumului de vizualizare), atunci nodul respectiv şi toţi fiii lui sunt

invizibili şi sunt părăsiți. Funcția `Invisible()` a clasei `Node` returnează 1, dacă volumul de delimitare al nodului este complet în afara volumului de vizualizare, şi 0 în celelalte cazuri.

Din clasa `Node` sunt derivate mai multe tipuri de clase, dintre care cele mai importante sunt: clasa `Material`, care defineşte aspectul obiectelor, clasa `Shape`, care defineşte forma obiectelor, clasa `Group`, care grupează mai multe obiecte, etc.

Reprezentarea aspectului obiectelor. Pentru descrierea aspectului obiectelor se defineşte materialul şi caracteristicile de texturare. În proiectul CSV s-a introdus doar proprietatea de material. Proiectul poate fi extins cu uşurință şi pentru texturare.

Materialul este descris de clasa `Material`, care defineşte coeficienții de reflectanță ambientală, difuză şi specular ai unui material dat printr-un nume unic în program.

```
class Material : public Object{
   char* name;                    // nume material
   GLfloat diffuse[4];            // componenta de difuzie
   GLfloat ambient[4];            // componenta ambientală
   GLfloat specular[4];           // componenta specular
   GLfloat shininess;             // exponent specular
public:
   Material(char* n = ""){        // material implicit
      name = new char(strlen(n) + 1);
      strcpy(name, n);
      for (int i=0; i<4;i++){
         diffuse[i] = 1.0;
         ambient[i] = 0.0;
         specular[i]= 1.0;
         shininess = 20;
      }
   }
   Material(Material &m);
   ~Material(){delete name;}
   char* GetName(){return name;}
   float* GetAmbient(){return ambient;}
   float* GetDiffuse() {return diffuse;}
   float* GetSpecular() {return specular;}
   float GetShininess() {return shininess;}
};
```

Un obiect din clasa `Material` (de fapt, un pointer la un astfel de obiect), este folosit ca atribut al unei mulțimi de fețe ale unui obiect (`IndexedFaceSet`). La întâlnirea unui obiect din clasa `Material` în cursul traversării bazei de date, materialul respectiv este setat ca material curent al bibliotecii de redare (în mod obişnuit OpenGL).

Reprezentarea formei obiectelor. Modul cel mai obişnuit de reprezentare a obiectelor tridimensionale în scenele virtuale este reprezentarea poligonală. În exemplul implementat, un astfel de obiect tridimensional este descris de clasa Shape, derivată din clasa Node şi conţine listele de vârfuri, normale, coordonate de texturare, feţe şi încă mai multe alte informaţii.

```
class Shape : public Node{
   ObArray coords;
   ObArray normals;
   ObArray texCoords;
   ObArray colors;
   ObArray faceSet;
   Vect3 center;
public:
   Shape(){}
   Shape(double s, int numcoord, int numfaces,
         int numvert, int *f, double  *c,
         double* n=0, double* t=0, double* col=0);

Shape(char *fn);
ObArray* GetCoords()  {return &coords;}
ObArray* GetNormals() {return &normals;}
ObArray* GetTexture() {return &texCoords;}
ObArray* GetColors()  {return &colors;}
IndexedFaceSet* GetFaceSet() {
   int index = faceSet.GetSize()-1;
   return (IndexedFaceSet*)faceSet.GetAt(index);
}
void AddCoord(Vect3* pVect3){
   coords.Add(pVect3);
   center+=*pVect3;
}
void AddFaceSet(IndexedFaceSet* pSet){
   faceSet.Add(pSet);
}
   void ReadObj(char *fileName);
   void Draw();
};
```

Clasa `Shape` descrie o reţea de poligoane care poate reprezenta o suprafaţă de frontieră închisă sau deschisă a unui solid. Impunerea condiţiilor de închidere sau orientare consistentă a suprafeţei reprezentate prin reţeaua de poligoane depinde de scopul aplicaţiei şi poate fi o funcţie de verificare inclusă în constructorul clasei `Shape`.

Vectorii de coordonate ale vârfurilor (`coords`) şi de normale (`normals`) sunt compuşi din pointeri la obiecte de tip `Vect3`. Vectorul de coordonate de texturare (`texCoords`) este un vector de pointeri la clasa `Vect2`, iar vectorul de culori (`colors`) este un vector de pointeri la clasa `Vect4`. Feţele unui obiect

Shape sunt reprezentate printr-un vector (faceSet) de pointeri la clasa IndexedFaceSet. Clasa IndexedFaceSet descrie o mulţime de feţe ale unui obiect, grupate datorită unei proprietăţi comune (de exemplu, material comun).

```
class IndexedFaceSet:public Object{
   int mode;              // mod de redare
   Material *material; // materialul
   ObArray faces;         // lista de feţe
public:
   IndexedFaceSet(
      material = NULL;
   }
   IndexedFaceSet(char *matName);
   void AddFace(IndexedFace* pFace){
      faces.Add(pFace);
   }
   void Draw(ObArray *coords, ObArray *normals,
            ObArray *texCoords, ObArray *colors);
   ..........................................................
};
```

O faţă din lista de feţe (faces) este descrisă de clasa IndexedFace, care conţine listele de indici reprezentate ca vectori de numere întregi (din clasa IntArray).

```
class IndexedFace : public Object{
   IntArray coordIndex;     // indici de coordonate
   IntArray normalIndex;    // indici de normale
   IntArray texCoordIndex; // indici de coord. textura
   IntArray colorIndex;     // indici de culoare
public:
   IndexedFace(){ mode = MODE_NONE;}
   void AddCoordIndex(int index){
      coordIndex.Add(index);
   }
   void SetMode(int m){ mode |=m;}
   void ResetMode(int m){mode &=~m;}
   int GetMode(){return mode;}
   void Draw(ObArray *coords, ObArray *normals,
         ObArray *texCoords, ObArray *colors);
   ..........................................................
};
```

O formă tridimensională (obiect din clasa Shape) se construieşte folosind unul din constructorii clasei Shape. Construcţia se realizează prin definirea conţinutului vectorilor de date (coordonate, normale, feţe, etc) prin citirea unor fişiere de un anumit tip (ReadObj(), pentru format *obj* al bazei de date, ReadVrl(), pentru fişiere VRML, etc.).

În fig. 11.2 este prezentată ierarhia claselor care descriu un obiect poligonal. Unele clase sunt descrise simplificat. De exemplu, vectorii `coords`, `normals`, `texCoords`, `colors` nu conţin direct coordonatele ci pointeri la obiecte de tip `Vect`, care conţin coordonatele respective. De asemenea, vectorii de indici pot conţine mai mult de trei indici în cazul general.

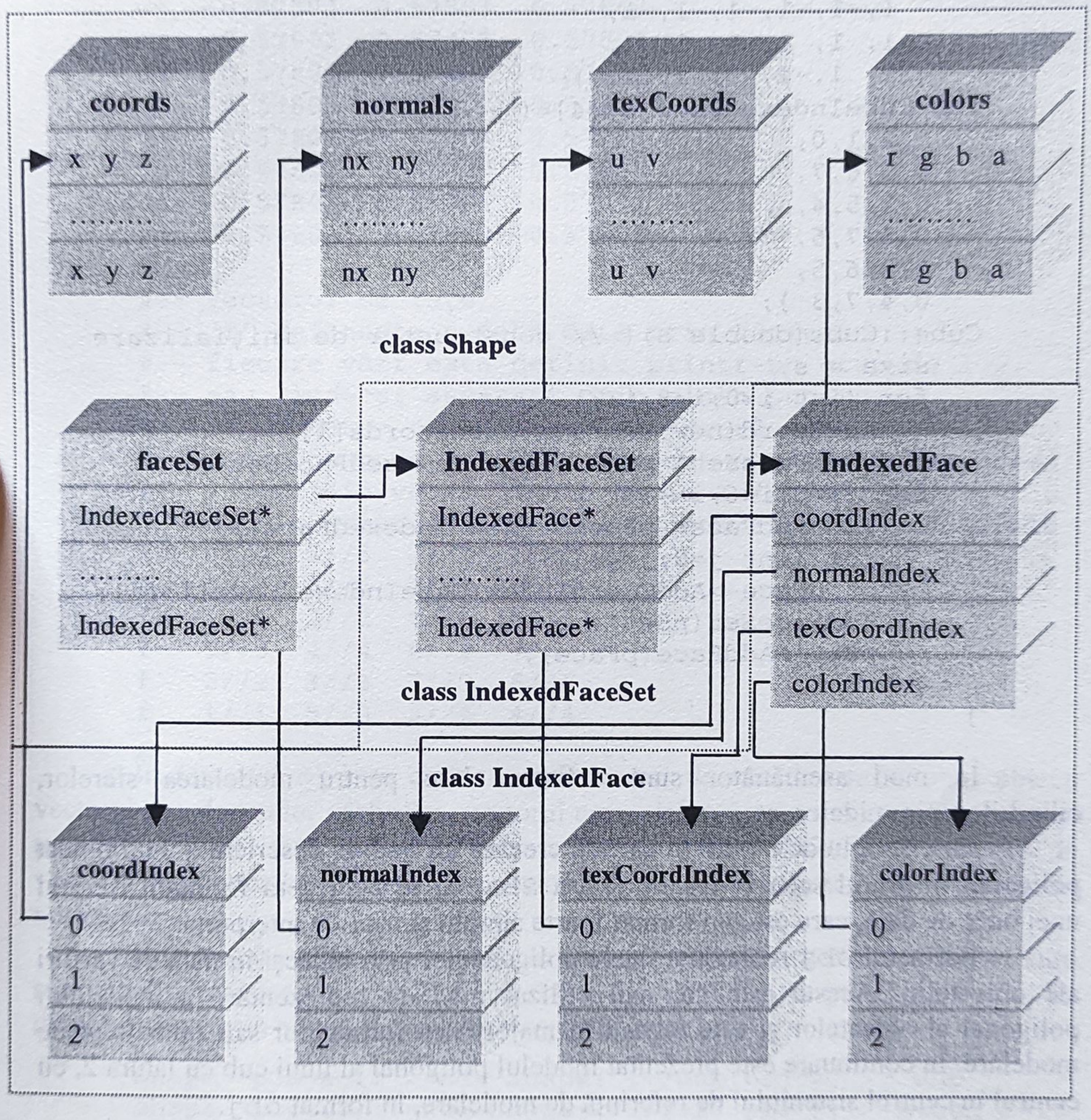


Fig. 11.2 Clasele de reprezentare a unui obiect poligonal: `Shape`, `IndexedFaceSet` şi `IndexedFace`.

Din clasa `Shape` sunt derivate clase care descriu forme geometrice particulare putând fi generate direct sau prin calcul. De exemplu, clasa `Cube` modelează un cub printr-o mulţime de feţe indexate.

```
class Cube : public Shape{
   double size;
public:
   Cube(double s = 1.0);
};
double CubeCoords[8][3]={
     -1,-1, 1, 1,-1, 1,
      1,-1,-1,-1,-1,-1,
     -1, 1, 1, 1, 1, 1,
      1, 1,-1,-1, 1,-1, };
int CubeIndexedFace[6][4]={
   3,2,1,0,
   4,5,6,7,
   0,1,5,4,
   2,3,7,6,
   1,2,6,5,
   0,4,7,3 };
Cube::Cube(double s){ // constructor de iniţializare
   size = s;
   for (int i=0;i<8;i++)
      AddCoord(new Vect3(s,CubeCoords[i]));
   IndexedFaceSet* pSet = new IndexedFaceSet();
   for (i=0;i<6;i++){
      IndexedFace* pFace = new IndexedFace();
      for (int j=0;j<4;j++)
         pFace->AddCoordIndex(CubeIndexedFace[i][j]);
      AddFaceSet(pSet);
      pSet->AddFace(pFace);
   }
}
```

În mod asemănător sunt definite clase pentru modelarea sferelor, cilindrilor, piramidelor, etc.

Ca exemplu de modul în care se creează un nod de descriere a unui obiect poligonal în graful scenei virtuale (clasa `Shape`) se consideră formatul `obj` al unei baze de date, care este un format foarte simplu şi uşor de interpretat.

Formatul `obj` reprezintă feţele poligoanelor prin indecşi în lista de vârfuri ale obiectului. Acesta este cel mai utilizat mod de reprezentare a modelului poligonal al obiectelor şi este regăsit în majoritatea formatelor sau limbajelor de modelare. În continuare este prezentat modelul poligonal al unui cub cu latura 2, cu centrul în centrul sistemului de referinţă de modelare, în format `obj`.

```
#--  Modelul unui cub în format obj-- Fişier cub.obj
#--  Vectorul de vârfuri
#--  'v' defineste un vârf
v   -1.00000 -1.00000  1.00000
v    1.00000 -1.00000  1.00000
v    1.00000 -1.00000 -1.00000
v   -1.00000 -1.00000 -1.00000
```

```
v   -1.00000  1.00000  1.00000
v    1.00000  1.00000  1.00000
v    1.00000  1.00000 -1.00000
v   -1.00000  1.00000 -1.00000
#--
#--  Vectorul de normale
#--  'vn' defineşte normala într-un vârf
vn  -0.57803 -0.57803  0.57803
vn   0.57803 -0.57803  0.57803
vn   0.57803 -0.57803 -0.57803
vn  -0.57803 -0.57803 -0.57803
vn  -0.57803  0.57803  0.57803
vn   0.57803  0.57803  0.57803
vn   0.57803  0.57803 -0.57803
vn  -0.57803  0.57803 -0.57803
#--
#-- Vectorul de feţe
#-- 'f' defineşte o faţă dată prin vârfuri
#-- fiecare vârf este definit printr-un grup de 1,2,
#-- sau 3indecsi separaţi prin slash pentru
#-- coordonate, coordonate de textură şi normală
#-- Coordonata de textură şi (sau) normala poate lipsi
#-- Indicii în vector încep de la 1 (nu de la 0)
#-- Feţele urmatoare au date coordonatele şi normalele
f   4//4  3//3  2//2  1//1
f   5//5  6//6  7//7  8//8
f   1//1  2//2  6//6  5//5
f   3//3  4//4  8//8  7//7
f   2//2  3//3  7//7  6//6
f   1//1  5//5  8//8  4//4
```

În formatul `obj` se definesc trei vectori de coordonate ai unui obiect: vectorul coordonatelor vârfurilor, vectorul normalelor şi vectorul coordonatelor de texturare. Fiecare faţă poligonală este o listă de vârfuri definite prin indecşi în vectorul coordonatelor vârfurilor, în vectorul coordonatelor de texură şi în vectorul normalelor.

Prin citirea unui fişier `obj` se creează un nod de tipul `Shape` care reprezintă descrierea poligonală a obiectului. De exemplu, se pot creea noduri cu diferte obiecte prin citirea fişierelor corespunzătoare:

```
Shape *pObject1 = new Shape("../cub.obj"); // cub
Shape *pObject1 =new Shape("../f-16.obj");// avion F-16
```

Reprezentarea nodurilor de grupare. Nodurile de grupare sunt nodurile care admit unul sau mai multe noduri fii în graful scenei. Clasa de bază pentru toate nodurile de grupare este clasa `Group`, derivată din clasa `Node` şi din ea sunt derivate clasele `Transform`, `LOD`, `Switch`, `Billboard`.

Clasa `Group` definește un vector de pointeri la alte noduri, care devin astfel noduri fii ale nodului respectiv:

```
class Group : public Node{
   ObArray childs;
public:
   void AddChild(Node* pNode){
      childs.Add(pNode);
   }
   Node* GetChild(int i){
      return (Node*)childs.GetAt(i);
   }
   int GetSize(){return childs.GetSize();}
   void Draw();
};
```

Clasa `Transform` este derivată din clasa `Group` și permite introducerea unei transformări printr-o matrice de transformare care se aplică tuturor fiilor acestui nod:

```
class Transform : public Group{
   double matrix[16];
public:
   Transform();                    // constructor implicit
   Transform(double *m);           // inițializare
   void Scale(double x, double y, double z);
   void Rotate(double angl,double x,double y,double z);
   void Translate(double x, double y, double z);
   void Draw();
};
```

Matricea de transformare `matrix` memorează o matrice 4×4 în modul coloană majoră. Ea este inițializată cu matricea identitate de către constructorul implicit al clasei și poate fi modificată prin funcțiile membre de translație, rotație, scalare.

Clasa `LOD` (*level of detail*) introduce un obiect reprezentat cu nivele de detaliu multiple. Modelarea cu nivele de detaliu multiple și selecția unui nivel la redare reprezintă una din cele mai importante metode de menținere a unei viteze constante de generare a imaginii scenelor virtuale complexe. Orice sistem grafic are o capacitate finită de redare a primitivelor grafice și menținere a unei frecvențe constante a cadrelor imaginii (număr de cadre/secundă) necesită menținerea unui număr cât mai redus de poligoane de reprezentare a obiectelor, asigurând în același timp realismul vizual dorit.

Posibilitatea prelucrării cu nivele de detaliu multiple a obiectelor este dată de caracteristica proiecției perspectivă, conform căreia dimensiunea proiecției unui obiect scade odată cu creșterea distanței acestuia față de punctul de observare (z_V). Acest lucru se poate observa din relația (4.10) care este reprodusă în continuare:

$$x_N = \frac{d\, x_V}{g\, z_V}, \quad y_N = \frac{d\, y_V}{h\, z_V}$$

Detaliile obiectelor aflate la distanţă mare faţă de punctul de observare nu mai sunt vizibile (imaginea lor putând ajunge sub dimensiunea unui pixel), de aceea aceste obiecte se pot reprezenta cu un număr mai mic de poligoane. Prelucrarea cu nivele de detaliu multiple constă în modelarea unui obiect printr-o succesiune de reprezentări din ce în ce mai simple (cu precizie şi număr de poligoane scăzut) şi selectarea versiuni de reprezentare în funcţie de distanţă.

Nivelele de detaliu ale unui obiect se organizează ca un grup de reprezentări, şi, pentru fiecare dintre ele se specifică intervalul de distanţe între care se va selecta la redare. În cursul redării imaginii se calculează distanţa de la punctul de observare la centrul obiectului şi, în funcţie de aceasta, se selectează reprezentarea adecvată.

O astfel de prelucrare este implementată printr-un nod în graful scenei de tipul `LOD`, care este derivat din clasa `Group`, deci admite un număr oarecare de noduri fii, fiecare nod fiu fiind o reprezentare pe un anumit nivel de detaliu. Clasa `LOD` conţine un vector de distanţe (`range`), de dimensiune egală cu numărul de nivele de detaliu (nodurile fii). Fiecare valoare din vectorul de distanţe reprezintă un punct de comutare între nivele. Nivelul de detaliu 0 (cu precizia maximă) este vizibil de la distanţa 0 până la distanţa memorată în primul element al vectorului de distanţe (`range[0]`); celelalte nivele sunt vizibile între două valori consecutive memorate în vectorul de distanţe; ultimul nivel este vizibil până la o distanţă maximă (ultima valoare din vectorul `range`); peste această distanţă obiectul nu mai este vizibil.

```
class LOD : public Group {
   Vect3 center;
   DoubleArray range;
public:
   LOD():center(0.0,0.0,0.0){}
   LOD(int n, double* r);
   double Dist();
   void Draw();
};
LOD::LOD(int n, double *r):center(0.0,0.0,0.0){}{
   for (int i=0;i<n;i++){
      range.Add(r[i]);
   }
}
```

La sfârşitul acestui subcapitol sunt date mai multe exemple de construire şi redare a scenelor virtuale, inclusiv prelucrarea nivelelor de detaliu multiple.

În fig. 11.3 este prezentată ierarhia principalelor clase de reprezentare a nodurilor grafului scenei virtuale în proiectul CSV. Clasa `Light` introduce o sursă de lumină în graful scenei.

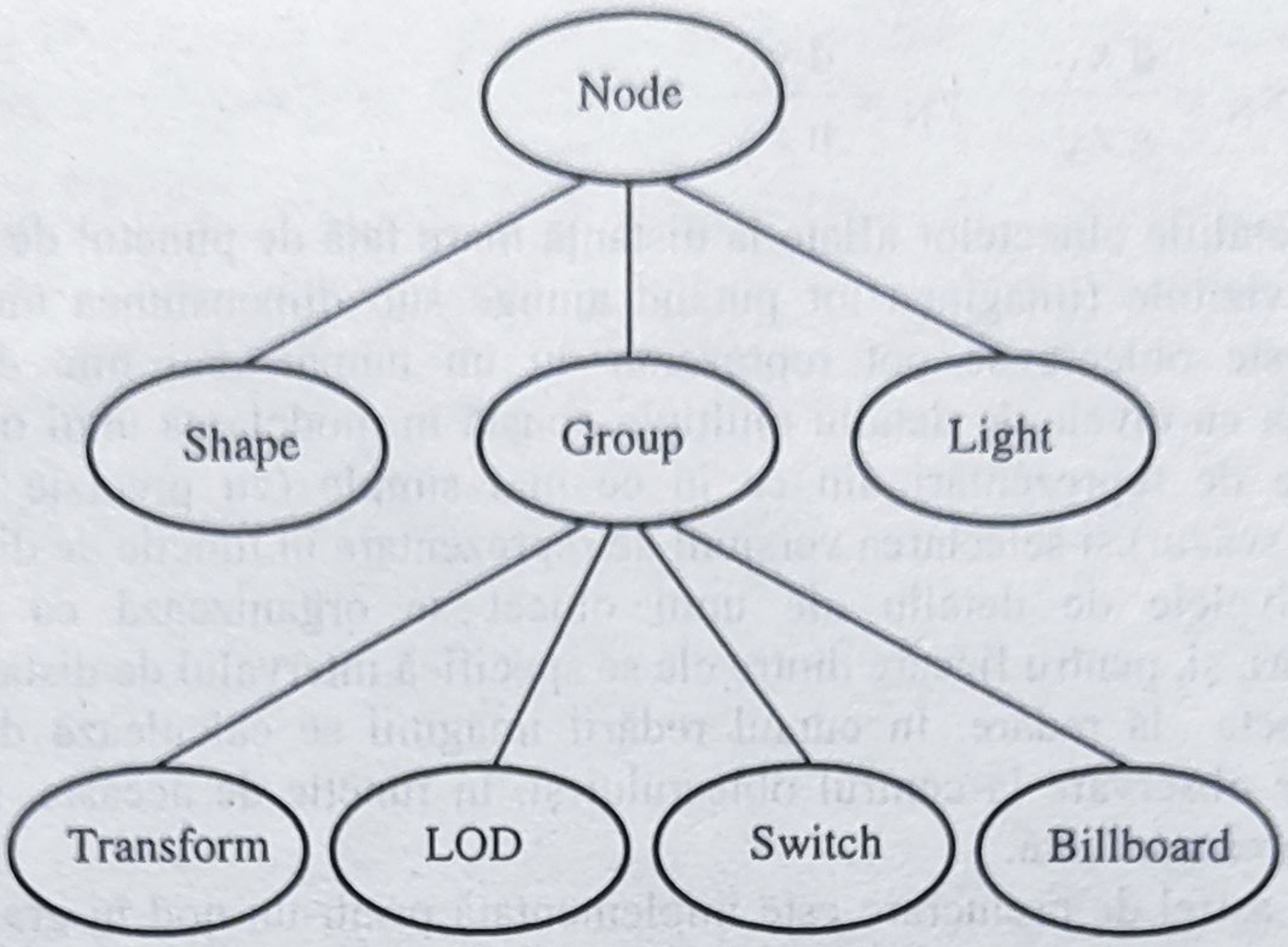


Fig. 11.3 Ierarhia claselor de descriere a grafului scenei.

11.2.2 REDAREA SCENELOR VIRTUALE

Graful de reprezentare a unei scene virtuale se creează prin citirea unuia sau mai multor fişiere de date. În exemplele din text s-au folosit fişiere în format `obj`, dar modelul în format intern al scenei (graful scenei) este acelaşi, indiferent de formatul bazei de date încărcate.

În cursul operaţiei de parcurgere în adâncime (traversare) a grafului scenei, în fiecare nod vizitat se apelează funcţia de redare a nodului, care în proiectul CSV este numită funcţia `Draw()`. Această funcţie testează mai întâi vizibilitatea volumului de delimitare şi, dacă acesta este invizibil funcţia se termină şi nodul împreună cu fii lui sunt ignoraţi (operaţie care se numeşte "retezarea" nodului). Dacă nodul nu este retezat, se apelează recursiv funcţiile de redare a nodurilor descendente şi, pentru unele din tipurile de noduri de grupare, se actualizează matricea curentă a stivei matricelor de transformări, pentru instanţierea (plasarea) obiectelor în sistemul de coordonate universal.

Un obiect tridimensional reprezentat poligonal prin clasa `Shape` este întotdeauna nod frunză al grafului, deoarece nu poate conţine noduri fii. Pentru redarea imaginii unui astfel de obiect se apelează funcţia `Draw()`, membră a clasei `Shape`. Funcţia `Draw()` a clasei `Shape` apelează funcţia membră `Draw()` a clasei `IndexedFaceSet` pentru fiecare mulţime de feţe indexate. Aceasta, la rândul ei, apelează funcţia membră `Draw()` a clasei `IndexedFace` pentru fiecare faţă. Funcţia `Draw()` a unei feţe (clasa `IndexedFace`) conţine toate apelurile de funcţii OpenGL, ca `glVertex#()`, `glNormal#()`, `glTexCoordinate#()`, depinzând de modul de redare a obiectului. Funcţiile `Draw()` ale claselor de reprezentare a unei forme poligonale arată astfel:

```
void Shape::Draw(){
   if(Invisible())    // testare volum de delimitare
      return;         // retezare (culling)
   int size = faceSet.GetSize();
   for (int i=0;i<size;i++){
      IndexedFaceSet* pSet =
        (IndexedFaceSet*)faceSet.GetAt(i);
      pSet->Draw(&coords, &normals,
                 &texCoords, &colors);
   }
}
void IndexedFaceSet::Draw()(ObArray *coords,
     ObArray *normals, ObArray *texCoords,
     ObArray *colors){
   int size = faces.GetSize();
   if (material){
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
                   material->GetAmbient());
      glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,
                   material->GetDiffuse());
      glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
                   material->GetSpecular());
      glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS,
                   material->GetShininess());
   }
   for (int i=0;i<size;i++){
      IndexedFace* pFace= (IndexedFace*)faces.GetAt(i);
      pFace->Draw(coords,normals,texCoords,colors);
   }
}
void IndexedFace::Draw(ObArray *coords,
   ObArray*normals,ObArray*texCoords,ObArray *colors) {
   int size = coordIndex.GetSize();
   glBegin(GL_POLYGON);
   for (int i=0;i<size;i++){
      int   index = coordIndex.GetAt(i);
      Vect3* coord = (Vect3*)coords->GetAt(index);
      if (mode & MODE_SMOOTH){
         index = normalIndex.GetAt(i);
         Vect3* norm = (Vect3*)normals->GetAt(index);
         glNormal3dv(norm->Get());
      }
      if (mode & MODE_TEXTURE){
         index = texCoordIndex.GetAt(i);
         Vect2* tex = (Vect2*)texCoords->GetAt(index);
         glTexCoord2dv(tex->Get());
      }
      glVertex3dv(coord->Get());
   }
   glEnd();
}
```

La traversarea unui nod de tip `Group` trebuie să se asigure condiţii de instanţiere identice fiecărui nod fiu al grupului. De aceea, matricea de transformare curentă (din capul stivei `MODELVIEW`) trebuie să fie salvată înaintea traversării fiecărui fiu şi restaurată după traversarea acestuia. Funcţia membră `Draw()` a clasei `Group` este următoarea:

```
void Group::Draw(){
   if(Invisible())    // testare volum de delimitare
      return;         // retezare (culling)
   int size=childs.GetSize();
   for (int i=0;i<size;i++){
      glPushMatrix();
         GetChild(i)->Draw();
      glPopMatrix();
   }
}
```

Clasa `Transform` redă mulţimea de noduri fii după aplicarea transformării definite de matricea nodului, prin înmulţirea acesteia cu matricea curentă din stiva matricelor de modelare-vizualizare. Funcţia `Draw()` a clasei `Transform` este:

```
void Transform::Draw(){
   glMultMatrix(matrix);
   Group::Draw();
}
```

La traversarea unui nod din clasa `LOD` se selectează nodul fiu corespunzător distanţei centrului obiectului faţă de punctul de observare. Funcţia `Draw()` a clasei `LOD` realizează acest lucru:

```
void LOD::Draw(){
   double d = Dist();    // calculează distanţa
                         // fata de punctul de observare
   int size = range.GetSize();
   for (int i=0;i<size;i++){
      if (d <=range[i])
         break;
   }
   if (i <size){
      printf("LOD:%d  Dist:%f\n",i,d);
      glPushMatrix();
         GetChild(i)->Draw();
      glPopMatrix();
   }
}
```

Toate aceste funcţii de redare (funcţiile `Draw()`) ale claselor nodurilor din graful scenei implementează algoritmul de traversare a grafului scenei prin parcurgerea în adâncime a grafului combinată cu actualizarea stării matricei

curente de transformare, prezentat principial la începutul acestui subcapitol. Selecţia operaţiilor executate în funcţie de tipul nodului se face prin faptul că funcţiile de redare sunt funcţii membre ale diferitelor clase de reprezentare a nodurilor.

În continuare sunt prezentate câteva exemple de construire şi redare a obiectelor şi scenelor virtuale ierarhice.

■ Exemplul 11.1

În acest exemplu se creează modelul poligonal al diferitelor obiecte prin citirea unor fişiere de date în format `obj`. Funcţia `Display()` a programului apelează funcţia de redare `Draw()` a nodului rădăcină al grafului scenei (`pNode`).

```
#include "../Object.h"
static Node* pNode; //nodul rădăcină al grafului scenei

Node* BuildScene1(){
    Shape *pShape;
    pShape = new Shape("../rose.obj");  // model vaza
//pShape = new Shape("../porsche.obj");// model masina
//pShape = new Shape("../flowers.obj");// model flori
    return pShape;
}
void Init(){
    pNode = BuildScene1();
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
}
void Display(){
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
        glRotated(-45, 0,1,0);
        glTranslated(-50,-10,-50);
        glScaled(0.3,0.3,0.3);
        pNode->Draw();
    glPopMatrix();
    glutSwapBuffers();
}
```

Funcţiile `Reshape()` şi `main()` sunt aceleaşi ca în programele precedente. În funcţia `Init()` este apelată funcţia de construcţie a scenei virtuale `BuildScene1()`. În acest exemplu scena constă dintr-un singur nod de tipul `Shape`, care reprezintă un obiect poligonal a cărui descriere se încarcă dintr-un

fişier de date în format `obj`. La execuţia acestui program se afişează imaginea obiectului încărcat: o vază cu un trandafir, o maşină Porsche sau un ghiveci cu flori (fig. 11.4).

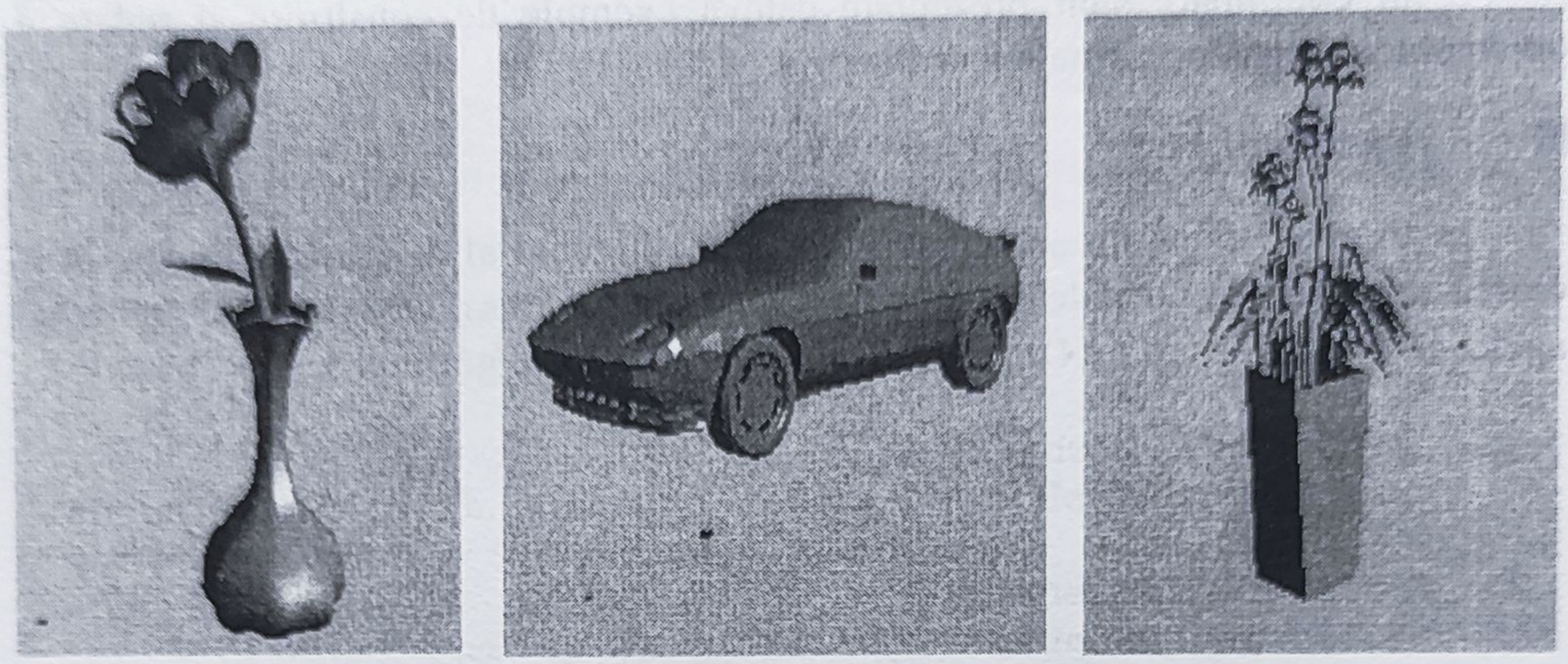


Fig. 11.4 Redarea obiectelor poligonale: (a) rose.obj (2246 poligoane); (b) porsche.obj (4740 poligoane); (c) flowers.obj (4061 poligoane).

Scene virtuale complexe, compuse din numeroase obiecte, se pot construi prin conectarea în graf a diferitelor tipuri de noduri. În general, structura grafului se creează prin citirea fişierelor bazei de date reprezentată într-un format care descrie organizarea ierarhică a scenei, cum sunt formatele `flt`, `dwb` sau limbajul VRML. În proiectul simplu dezvoltat în acest capitol, graful scenei virtuale poate fi creat prin program, ca în exemplele următoare.

■ **Exemplul 11.2**

În acest exemplu se construieşte o scenă compusă din patru obiecte avioane F-16, plasate în diferite poziţii în scenă. În funcţia `Init()` se apelează o altă funcţie de construcţie a scenei, `BuildScene2()`, care creează un graf al scenei compus din patru obiecte F-16 amplasate în diferite poziţii în scenă astfel:

```
Node* BuildScene2(){
   Group *pG = new Group;
   Shape *pO = new Shape("../f-16.obj");
   Transform* pT;
   // Prima instantiere
   pT = new Transform;
   pT->AddChild(pO);
   pG->AddChild(pT);
   // A doua instantiere
   pT = new Transform;
   pT->Translate(0.0, 8.0, -40.0);
   pT->Rotate(20.0, -90.0, -10.0);
```

```
    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    // A treia instantiere
    pT = new Transform;
    pT->Translate(30.0,16.0,-40.0);
    pT->Rotate(20.0,90.0,45.0);
    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    // A patra instantiere
    pT = new Transform;
    pT->Translate(-30.0,16.0,-40.0);
    pT->Rotate(20.0,-90.0,-45.0);
    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    return pG;
}
```

Graful scenei create în funcţia `BuildScene2()` este prezentat în fig. 11.5.

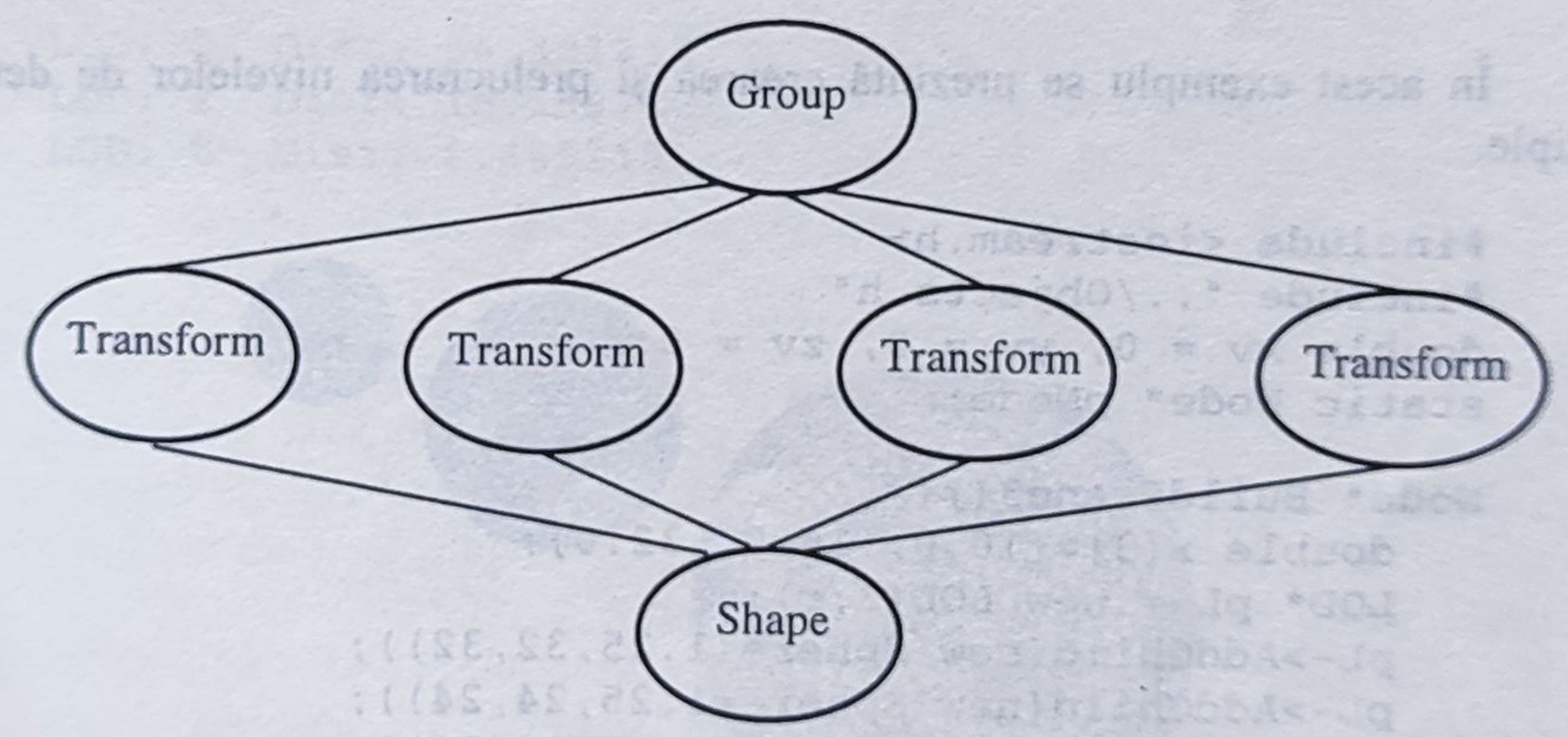


Fig. 11.5 Graful scenei virtuale construit de funcţia `BuildScene2()`.

Nodul rădăcină al grafului este un nod de tip `Group`, care are 4 noduri fii. Fiecare nod fiu este un nod de tip `Transform`, care defineşte poziţia sistemului de referinţă model în sistemul de referinţă universal. Fiecare nod de transformare are câte un singur nod descendent, şi anume nodul de tip `Shape` care modelează avionul F-16.

Imaginea afişată la execuţia programului este dată în fig. 11.6. Axele de coordonate ale sistemului universal sunt reprezentate prin apelul funcţiei `Axes()` (prezentată într-unul din programele precedente) în funcţia `Display()`.

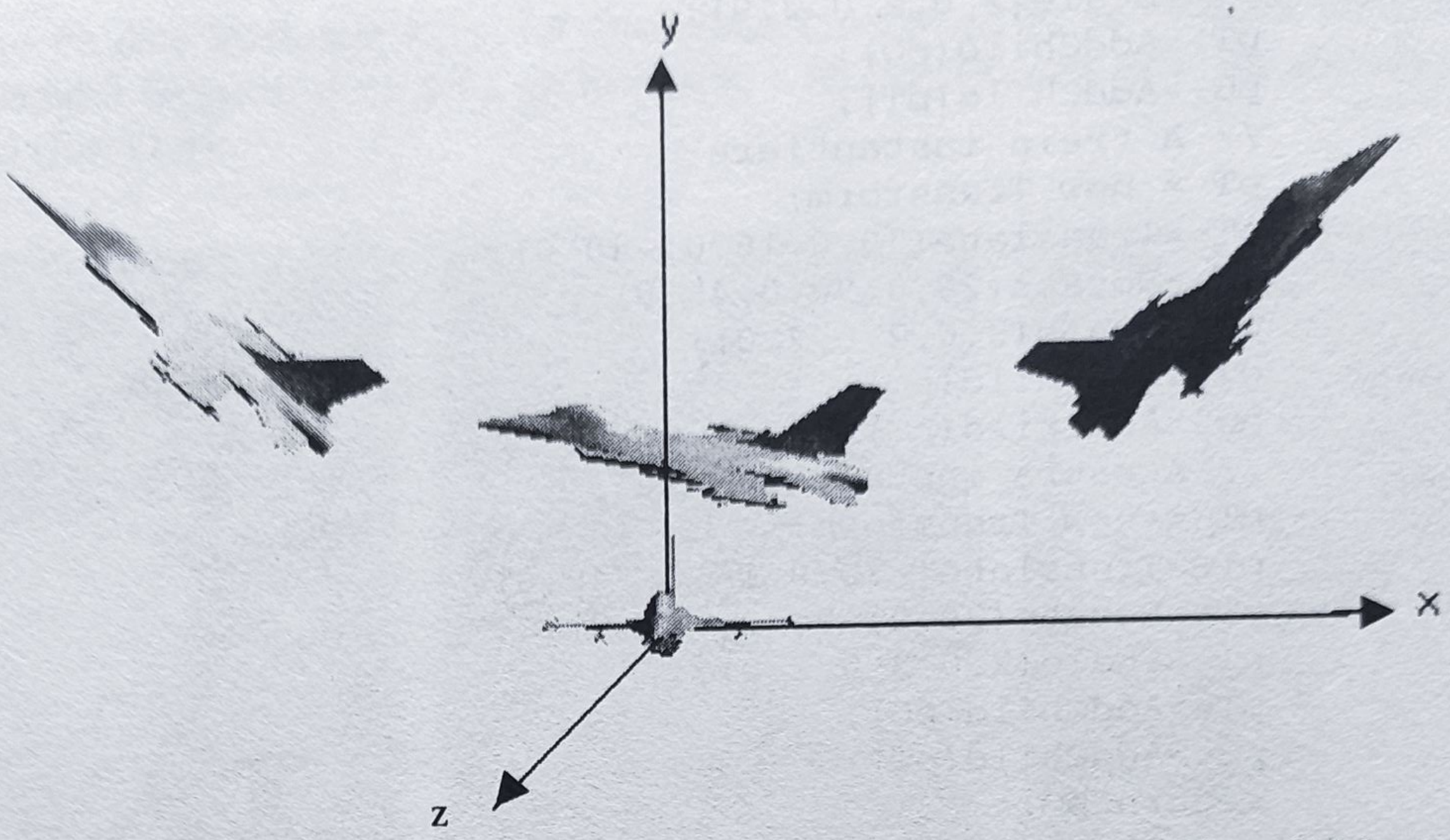


Fig. 11.6 Imaginea scenei virtuale create prin instanțierea a patru obiecte F-16.

■ Exemplul 11.3

În acest exemplu se prezintă crearea şi prelucrarea nivelelor de detaliu multiple.

```
#include <iostream.h>
#include "../Objects.h"
double xv = 0, yv = 0, zv = 0;
static Node* pNode;

Node* BuildScene3(){
   double r[3]={10.0, 16.0, 32.0};
   LOD* pL = new LOD(3,r);
   pL->AddChild(new Sphere(1.25,32,32));
   pL->AddChild(new Sphere(1.25,24,24));
   pL->AddChild(new Sphere(1.25,14,14));

   Group *pG = new Group;
   Transform* pT = new Transform;
   pT->Translate(-4,-2,-26);
   pT->AddChild(pL);
   pG->AddChild(pT);
   pT = new Transform;
   pT->Translate(0,-2,-15);
   pT->AddChild(pL);
   pG->AddChild(pT);
   pT = new Transform;
   pT->Translate(2,-2,-8);
   pT->AddChild(pL);
```

```
        pG->AddChild(pT);
        return pG;
    }
    void Init(){
        pNode = BuildScene3();
        glColor3f(0.2,0.2,0.2);
    }
     void Display(){
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        pNode->Draw();
        glutSwapBuffers();
    }
```

În funcția `BuildScene3()` se creează graful scenei compus din trei instanțe ale unui obiect cu trei nivele de detaliu. Obiectul este o sferă şi pentru fiecare nivel de detaliu se reprezintă printr-un număr diferit de fețe: nivelul 0 este reprezentat prin $32 \times 32 = 1024$ de fețe, nivelul 1 prin $24 \times 24 = 576$ fețe, nivelul 2 prin $14 \times 14 = 196$ fețe. Distanțele de comutare între nivelele de detaliu sunt date în vectorul `range`. Selecția nivelului de detaliu se face în funcție de distanța centrului sferei față de punctul de observare. La execuția acestui program se obține imaginea din fig. 11.7 şi se afişează la consolă mesajele:

```
LOD: 2  Dist: 26.381812
LOD: 1  Dist: 15.152746
LOD: 0  Dist: 8.485218
```

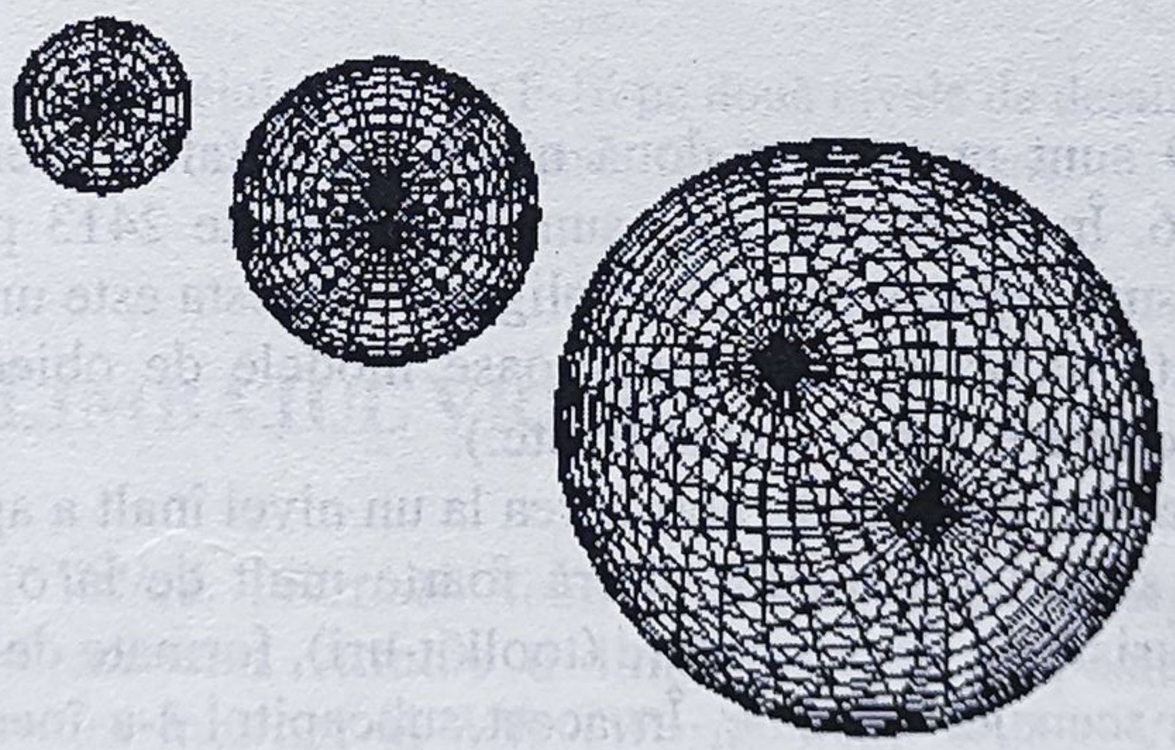


Fig. 11.7 Reprezentarea obiectelor cu nivele de detaliu multiple.

Din figura de mai sus reiese eficiența reprezentării obiectelor cu nivele de detaliu multiple. Sfera din stânga, aflată la distanța de 26.381, este redată pe nivelul 2 de detaliu, cu numai 196 de fețe. Sfera din dreapta, aflată la distanța de 8.485, este redată pe nivelul 0 de detaliu, cu 1024 de fețe. Şi, cu toată diferența de precizie a aproximării poligonale, aspectul nivelului 2 de detaliu nu este inferior nivelului 0, datorită dimensiunii reduse a obiectului în proiecție perspectivă, atunci când distanța este mai mare. Reprezentarea obiectelor aflate la distanță mare cu precizie

ridicată produce un consum inutil de timp de calcul pentru generarea imaginii. Ca exercițiu, se poate încerca reprezentarea sferelor cu aceeaşi precizie indiferent de distanța față de punctul de observare. Se va observa că reprezentarea tuturor obiectelor cu precizie maximă nu aduce nici un beneficiu din punct de vedere al aspectului pentru obiectele depărtate, iar reprezentarea cu precizie minimă este inaceptabilă pentru obiectele apropiate. Reprezentarea cu nivele de detaliu multiple este o modalitate mult mai eficientă din punct de vedere al timpului de execuție și păstrează (în anumite limite) realismul imaginii. Graful scenei create de funcția `BuildScene3()` este dat în fig. 11.8.

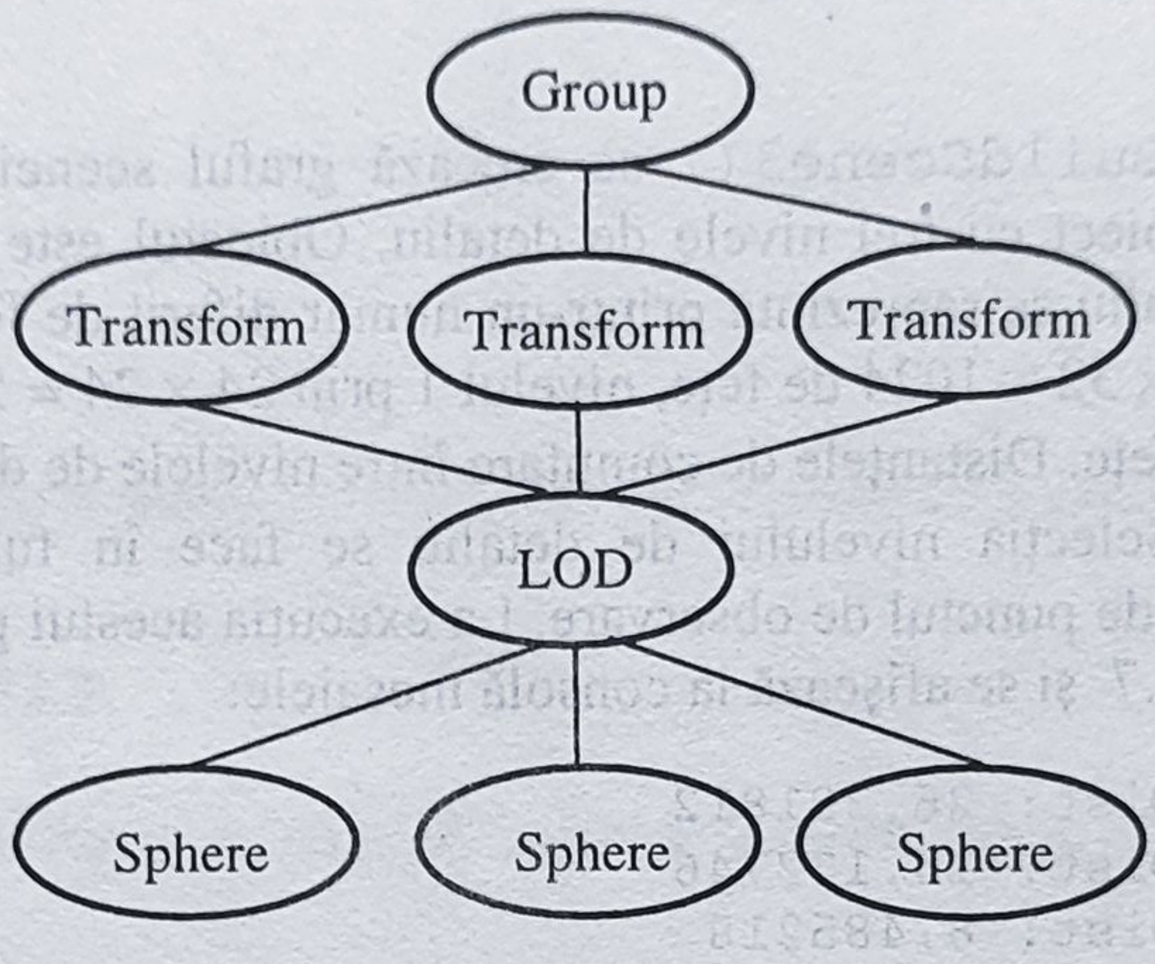


Fig. 11.8 Graful scenei create de funcția `BuildScene3()`.

În fig. 11.9 sunt prezentate două nivele de detaliu diferite ale aceluiaşi obiect, avionul F-16. În prima versiune sunt reprezentate 2413 poligoane; în cea de-a doua versiune sunt reprezentate 286 poligoane. Acesta este un model proiectat de firma Viewpoint, alături de alte numeroase modele de obiecte din cele mai variate domenii (elicoptere, maşini, clădiri, etc.).

Aşa cum s-a mai menționat, abordarea la un nivel înalt a aplicațiilor grafice (crearea şi redarea scenelor virtuale) diferă foarte mult de la o aplicație la alta, existând numeroase sisteme de dezvoltare (toolkit-uri), formate de baze de date sau limbaje de creare a scenelor virtuale. În acest subcapitol s-a încercat prezentarea unui proiect cât mai simplu de creare şi redare a scenelor virtuale, dar care înglobează majoritatea caracteristicilor sistemelor complexe, cum sunt Performer, WorldUp, dVS, EasyEcene, Multigen.

Este de aşteptat ca înțelegerea modalității principiale de creare şi redare a scenelor virtuale să ajute la abordarea şi înțelegerea rapidă a oricărui astfel de sistem de dezvoltare. De asemenea, un astfel mod de organizare ierarhică, printr-un graf aciclic direcționat, a scenelor virtuale, corespunde formatului unei baze de date grafice descrise în limbajul VRML. Acest lucru se va observa cu uşurință în subcapitolul următor.

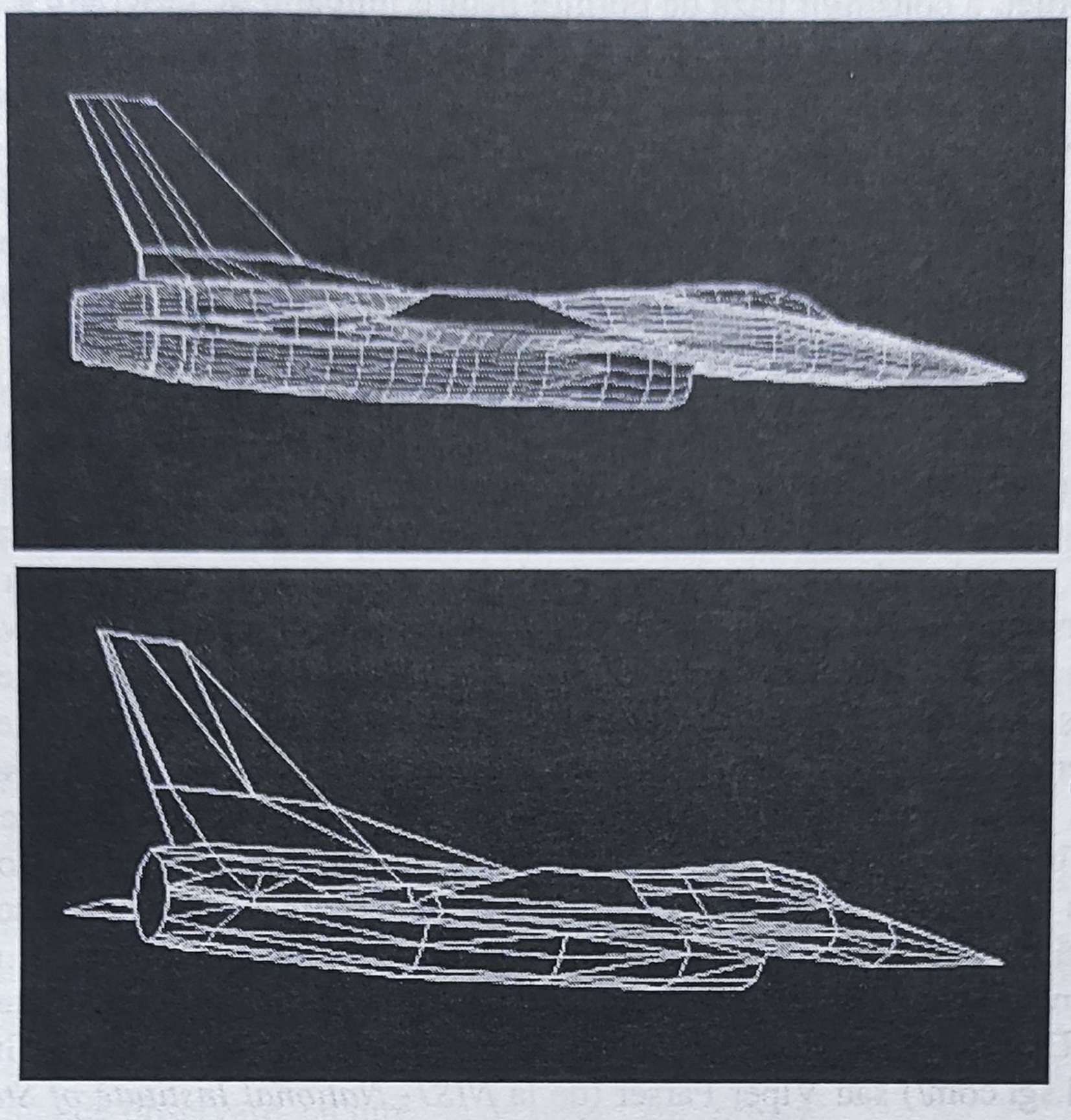


Fig. 11.9 Modelul avionului F-16 pe două nivele de detaliu diferite:
(a) 2413 poligoane; (b) 286 poligoane.

11.3 LIMBAJUL VRML

Limbajul *VRML (Virtual Reality Modeling Language)* este un limbaj folosit în crearea scenelor virtuale tridimensionale, cu posibilitatea de acces a acestora prin rețeaua Internet WWW (*World Wide Web*). Versiunea folosită în momentul de față este VRML 2.0 (numită și VRML 97, fiind standard ISO din anul 1997) și este o revizuire a versiunii VRML 1.0, stabilite în 1995. Numeroase informații referitoare la limbajul VRML se pot găsi în rețeaua Internet (http://www.vrml.org).

În prima versiune, limbajul VRML 1.0 permitea crearea unei scene virtuale compuse din obiecte tridimensionale, dar cu posibilități limitate de interactivitate prin rețeaua Internet. VRML 1.0 a fost dezvoltat pe baza formatului de descriere a bazelor de date Open Inventor, elaborat de firma Silicon Graphics, care suportă descrierea scenelor ierarhice cu obiecte poligonale, lumini, materiale și efecte realiste. Un subset al formatului Open Inventor, cu extensii pentru acces prin

reţeaua Internet, a constituit baza de pornire pentru limbajul de descriere a scenelor virtuale VRML. În momentul de faţă, numeroase titluri de cărţi, articole, reviste, documentaţie, exemple, adrese de grupuri de lucru, etc., se găsesc pe reţea despre limbajul VRML şi utilizarea acestuia.

Un fişier VRML este un fişier ASCII care conţine descrierea unei scenei virtuale şi a unor acţiuni interactive, şi poate fi accesat prin reţea folosind programe de navigare (browsere). Pentru a afişa o scenă VRML din reţeaua Internet sau dintr-un fişier de pe disc (cu extensia .vrl) este necesar un browser VRML, configurat ca o extensie (plug-in) al unui browser Web. Pentru browserul Netscape Navigator, există browserul Cosmo Player (de la compania Platinum Technology) pentru interpretarea fişierelor VRML. Un browser VRML citeşte fişierul care conţine descrierea scenei virtuale, creează graful scenei şi memorează datele de comportare a obiectelor şi de interacţiuni corespunzătoare. Observarea scenei virtuale şi navigarea în direcţia dorită se realizează prin comenzi ale browserului. Aceste comenzi sunt foarte simple şi se găsesc în documentaţia on-line a acestuia.

Deşi utilizarea cea mai frecventă a limbajului VRML este în crearea şi redarea scenelor 3D prin reţeaua Internet, un fişier VRML poate fi folosit şi ca descriere a unei scene virtuale pentru aplicaţii de realitate virtuală care nu se desfăşoară în reţea. În această situaţie, pentru citirea fişierelor VRML se foloseşte un program de interpretare a fişierelor VRML (*parser*) care creează graful scenei în concordanţă cu structura programului de aplicaţie. În domeniul public pe reţeaua Internet sunt disponibile un mare număr de parsere care pot fi integrate într-o aplicaţie. De exemplu, există VRML 2.0 Parser de la Silicon Graphics (http://vrml.sgi.com/) sau Viper Parser (de la *NIST- National Institute of Standard and Technology*) implementat în limbajul Java.

În oricare situaţie, formatul VRML permite construirea grafului scenei, furnizând toate informaţiile despre obiectele tridimensionale, ierarhia acestora, condiţii de mediu ambiant, materiale, etc. Graful scenei, construit prin citirea unuia sau mai multor fişiere VRML, este apoi traversat pentru obţinerea imaginii obiectelor în diferite situaţii de navigare a scenei. Redarea imaginii obiectelor pe display se realizează prin intermediul bibliotecilor grafice de nivel mai scăzut, cum sunt OpenGL sau Direct3D (de la Microsoft). Aceste biblioteci asigură interfaţa corespunzătoare cu acceleratorul grafic al sistemului.

11.3.1 SPECIFICAŢIILE LIMBAJULUI VRML

Specificaţiile limbajului VRML 97 descriu conceptele de realizare a scenelor virtuale, descrierea nodurilor, descrierea câmpurilor, a evenimetelor şi interacţiunea între noduri. Similar limbajelor de programare standard, specificaţiile VRML conţin reguli sintactice şi reguli semantice ale limbajului.

Un fişier VRML conţine un header (acelaşi pentru ISO VRML 97 şi VRML 2.0), urmat de descrierea nodurilor grafului scenei, care specifică formele geometrice, grupurile, transformările şi atributele mediului virtual ce se contruieşte şi vizualizează. Sintaxa VRML este simplă, cu cuvinte cheie în limba engleză sugestive pentru entitatea definită. Headerul fişierului este:

```
#VRML 2.0 utf8
# Alte comentarii
```

unde `utf8` este setul de caractere folosit. Este obligatorie introducerea versiunii `VRML 2.0` imediat după caracterul # (fără nici un spațiu liber).

Un nod este specificat prin tipul nodului, urmat de un bloc cuprins între acolade. Blocul din interiorul acoladelor conține zero sau mai multe câmpuri (*fields*) şi pentru fiecare câmp se specifică valoarea (sau valorile) acestuia (*field value*). Fiecare câmp al unui nod are un nume şi admite una sau mai multe valori de un tip de date determinat de tipul câmpului.

În VRML este extinsă noțiunea de nod pentru orice entitate care grupează mai multe informații. Se poate observa diferența dintre noțiunea de *nod VRML* şi noțiunea de nod într-un graf al scenei (cum au fost cele descrise în proiectul CSV). Un nod într-un graf este o componentă a unei structuri ierarhice, fiecare nod având unul sau mai mulți părinți (cu excepția nodului rădăcină care nu are nici un părinte) şi unul sau mai mulți fii (cu excepția nodurilor frunză care nu au fii). Un nod VRML poate fi nod în graful scenei, sau poate să fie inclus (ca valoare a unui câmp) într-un alt nod. Modul cum se construieşte graful scenei pornind de la specificarea nodurilor VRML depinde de programul de aplicație.

Sistemele de coordonate folosite în VRML sunt aceleaşi cu sistemle de coordonate din OpenGL (fig. 6.3): sistemul de referință universal este un sistem drept, sistemul de referință observator este de asemenea un sistem de referință drept, definit ca localizare şi orientare relativ la sistemul de referință universal; proiecția perspectivă se defineşte printr-un plan perpendicular pe axa z a sistemului de referință de observare, cu direcția de observare spre –z .

Tipurile de noduri VRML sunt următoarele:

- Forme şi descrieri geometrice (*shapes*): `Box`, `Cone`, `Coordinate`, `Cylinder`, `EvaluationGrid`, `Extrusion`, `IndexedFaceSet`, `IndexedLineSet`, `Normal`, `PointSet`, `Shape`, `Sphere`, `Text`.
- Aspect (*appearance*): `Appearance`, `Color`, `FontStyle`, `ImageTexture`, `Material`, `MovieTexture`, `PixelTexture`, `TextureCoordinate`, `TextureTransform`.
- Grupuri (*grouping*): `Anchor`, `Billboard`, `Collision`, `Group`, `Inline`, `LOD`, `Switch`, `Transform`.
- Mediu (*environment*): `AudioClip`, `Background`, `DirectionalLight`, `Fog`, `PointLight`, `Sound`, `SpotLight`.
- Vizualizare (*viewing*): `NavigationInfo`, `Viewpoint`.
- Animație (*animation*): `ColorInterpolator`, `CoordinateInterpolator`, `NormalInterpolator`, `OrientationInterpolator`, `PositionInterpolator`, `ScalarInterpolator`, `TimeSensor`.
- Interacțiuni (*interaction*): `CylinderSensor`, `PlaneSensor`, `ProximitySensor`, `SphereSensor`, `VisibilitySensor`, `TouchSensor`.
- Alte noduri: `Script`, `WorldInfo`.

În total sunt 54 de tipuri de noduri şi fiecare tip admite unul sau mai multe tipuri de câmpuri. Tipurile de date prin care se specifică valoarea unui câmp pot fi: numere întregi, numere în virgulă flotantă, valori boolene, şiruri de caractere, sau pot fi folosite chiar noduri VRML. Un exemplu de fişier VRML foarte simplu este dat în continuare.

■ **Exemplul 11.4**

```
#VRML 2.0 utf8
# Construieste un cilindru 3D
Shape {
   appearance Appearance {
      material Material {
         diffuseColor 0.0, 0.5, 1.0
      }
   }
   geometry Cylinder {
      height 2.0
      radius 1.5
   }
}
```

Nodul de tipul `Shape` introduce un obiect tridimensional pentru care se specifică câmpurile cu numele `appearance` şi `geometry`, pentru definirea aspectului şi a formei. Valoarea câmpului `appearance` este un nod de tipul `Appearance`, definit imediat prin blocul corespunzător între acolade. Nodul `Apperence` are specificat câmpul `material` a cărui valoare este dată printr-un nod de tipul `Material`. Nodul de tip `Material` conține câmpul `diffuseColor`, pentru care se specifică valoarea prin trei numere în virgulă flotantă, pentru componentele roşu, verde, albastru. Valoarea câmpului `geometry` este un nod de tipul `Cylinder`, care este definit prin câmpurile `heigth` şi `radius`, cu valori numere în virgulă flotantă.

Un fişier VRML poate conține mii de noduri şi fiecare nod conține un număr variabil de câmpuri, tipic două-trei câmpuri, dar pot fi şi zeci de câmpuri. Din fericire, toate câmpurile nodurilor au prevăzute valori implicite rezonabile, astfel că nodurile pot fi folosite chiar dacă nu se specifică valoarea tuturor câmpurilor. De exemplu, valoarea implicită a culorii de ştergere este negru şi, dacă nu se specifică altă culoare de fundal (printr-un nod `Background`), atunci la navigare prin intermediul browserelor fundalul rămâne negru. Poziția inițială a punctului de observare este la (0.0, 0.0, 10.0) şi, dacă nu se specifică altă poziție (printr-un nod `Viewpoint`), atunci navigarea începe cu această poziție.

Sunt considerate comentarii toate caracterele după caracterul # până la sfârşitul liniei. Nodurile, câmpurile şi valorile câmpurilor pot fi aranjate în oricâte linii; spațiile, tab-urile, virgulele şi caracterul de linie nouă sunt considerate spații albe (*white spaces*) şi ignorate la citire. VRML este case-senzitive, adică diferențiază literele mici de cele mari.

11.3.2 CONSTRUIREA SCENELOR VIRTUALE ÎN VRML

Scenele virtuale se construiesc din noduri de diferite tipuri (noduri de forme geometrice, noduri de grupare, noduri de transformări geometrice, etc.) conectate într-un graf aciclic direcţionat.

11.3.2.1 Construirea formelor geometrice

Orice formă geometrică se construieşte folosind un nod de tipul `Shape`, cu câmpurile `appearance` şi `geometry` pentru specificarea aspectului şi a formei geometrice.

Forma geometrică se specifică prin numele unuia din corpurile geometrice elementare recunoscute de VRML (`Box`, `Cylinder`, `Cone`, `Sphere`), printr-o grilă de valori de altitudini (`ElevationGrid`), printr-un obiect obţinut prin extrudare (`Extrusion`) sau printr-o reţea de feţe poligonale (`IndexedFaceSet`), linii (`IndexedLineSet`) sau puncte (`PointSet`).

Cea mai puternică şi mai folosită modalitate de reprezentare a obiectelor este reprezentarea poligonală descrisă de nodul `IndexedFaceSet`. Acest nod defineşte un obiect printr-o mulţime de vârfuri (coordonate 3D) şi o mulţime de feţe, fiecare faţă fiind o listă de indici în vectorul de vârfuri. Se poate urmări mai simplu definiţia unui astfel de nod în exemplul următor.

■ **Exemplul 11.5**

```
#VRML V2.0 utf8
# cub.wrl - construieşte un cub prin feţe indexate
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry IndexedFaceSet {
        coord Coordinate {
            point [
            # Coordonatele de deasupra a cubului
                -1.0  1.0  1.0,
                 1.0  1.0  1.0,
                 1.0  1.0 -1.0,
                -1.0  1.0 -1.0,
            # Coordonatele de jos ale cubului
                -1.0 -1.0  1.0,
                 1.0 -1.0  1.0,
                 1.0 -1.0 -1.0,
                -1.0 -1.0 -1.0
            ]
    }
    coordIndex [
        0, 1, 2, 3, -1,
        7, 6, 5, 4, -1,
        0, 4, 5, 1, -1,
```

```
                1, 5, 6, 2, -1,
                2, 6, 7, 3, -1,
                3, 7, 4, 0
            ]
        }
    }
```

Nodul `Shape` creează un cub şi specifică fiecare faţă poligonală printr-o listă de indici în lista de vârfuri. Orice vector (vectorul `point`, vectorul `coordIndex`) este încadrat între paranteze drepte. Delimitarea între indecşii feţelor se face prin numărul –1.

11.3.2.2 Nodurile de grupare

Nodurile de grupare (`Group`, `LOD`, `Switch`, `Billboard`, `Inline`, `Collision`, `Anchor`, `Transform`) ale limbajului VRML sunt noduri care au unul sau mai multe noduri fii.

Nodul `Group` colectează mai mulţi fii, care pot fi forme, lumini, sau alte noduri `Group`. În exemplul următor este creat un obiect compus din trei paralelipipede dreptunghice (semnul "plus" tridimensional).

■ **Exemplul 11.6**

```
#VRML V2.0 utf8
# semn3D.wrl - semnul "plus" tridimensional
Group {
    children [
        Background {
            skyColor  1.0 1.0 1.0
        },
        Viewpoint {
            position 40.0, 10.0, 40.0
            orientation 0.0 1.0 0.0 0.78
        },
        DirectionalLight {
            direction  0.0 -1.0 -0.2
            intensity 1.0
            ambientIntensity 0.2
            color 1.0 1.0 1.0
        },
        Shape {
            appearance DEF White Appearance {
                material Material { }
            }
            geometry Box {
                size 25.0 2.0 2.0
            }
        },
        Shape {
```

```
                appearance USE White
                geometry Box {
                    size 2.0 25.0 2.0
                }
            },
            Shape {
                appearance USE White
                geometry Box {
                    size 2.0 2.0 25.0
                }
            }
        ]
    }
```

Nodul `Group` conține șase noduri fii: trei noduri de descriere a unei forme geometrice (nodurile `Shape`), un nod de sursă lumină (`DirectionalLight`), un nod de definire a culorii de ștergere (`Background`) și un nod de poziționare a punctului de observare (`Viewpoint`). Lista de fii este introdusă între paranteze drepte precedate de cuvântul cheie `children`. În acest exemplu este folosită etichetarea nodurilor. Orice nod poate primi o etichetă introdusă prin cuvântul-cheie `DEF`. Un nod odată etichetat la definiție, poate fi folosit ca valoare a unui câmp prin simpla introducere a cuvântului-cheie `USE` urmat de eticheta nodului. În exemplul de mai sus, un nod de aspect a fost etichetat cu numele `White` (`DEF White Appearance{.....}`) și apoi folosit cu acest nume ca valoare a unui câmp (`appearance USE White`). Imaginea care se obține la vizualizarea acestui fișier VRML este dată în fig. 11.10.

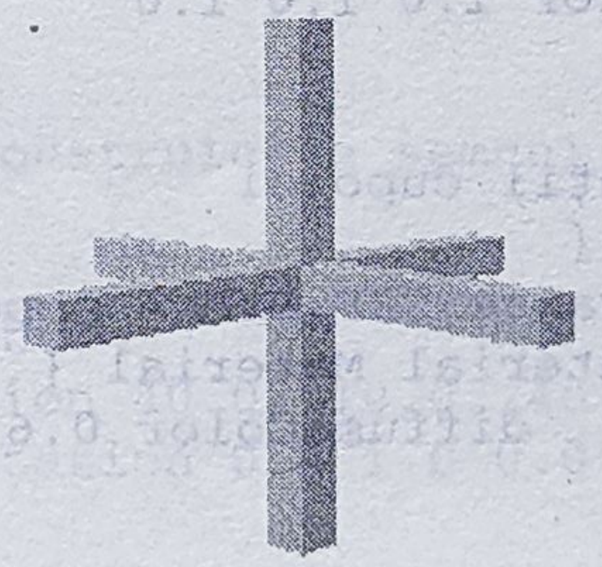


Fig. 11.10 Imaginea afișată de browserul Cosmo Player la încărcarea fișierului semn3D.wrl.

Această imagine se obține fie prin intermediul unui browser (Cosmo Player pentru Netscape), fie prin interpretarea (parsarea) fișierului, crearea grafului scenei și redarea imaginii acestuia.

În fișierul `semn3D.wrl` mai este folosit un nod pentru definirea culorii de fundal (nodul `Background` cu câmpul `skyColor`), nodul `Viewpoint` pentru definirea poziției de observare și nodul `DirectionalLight` pentru definirea unei surse de lumină direcțională. Se pot remarca unele diferențe între convențiile

de definire a unor date în VRML şi cele din OpenGL. De exemplu, pentru rotaţie se specifică mai întâi componentele axei de rotaţie, apoi unghiul de rotaţie în radiani (în OpenGL se dă mai întâi valoarea unghiului în grade, apoi componentele axei de rotaţie).

Nodul `Transform` defineşte transformarea de coordonate a tuturor nodurilor fii ai acestuia faţă de nodul părinte. Prin aplicarea acestei transformări, un nod fiu este transformat din sistemul de referinţă propriu în sistemul de referinţă al nodului părinte. Succesiunea de transformări care se aplică unui nod frunză îl transformă din sistemul de referinţă local în sistemul de referinţă universal.

■ **Exemplul 11.7**

Fişierul `cupola.vrl` de mai jos construieşte o cupolă din două obiecte poziţionate folosind un nod `Transform`:

```
#VRML V2.0 utf8
# cupola.wrl - construieşte o cupolă
Group {
    children [
        Background {
            skyColor  1.0 1.0 1.0
        },
        DirectionalLight {
            direction  0.0 -1.0 -1.0
            intensity 1.0
            ambientIntensity 0.2
            color 1.0 1.0 1.0
        },

        # Pereţii cupolei
        Shape {
        appearance DEF Brown Appearance {
            material Material {
                diffuseColor 0.6 0.4 0.0
            }
        }
        geometry Cylinder {
            height 2.0
            radius 2.0
        }
    },
    # Acoperişul cupolei
    Transform {
        translation 0.0 2.0 0.0
        children [
            Shape {
                appearance USE Brown
                geometry Cone {
```

de definire a unor date în VRML şi cele din OpenGL. De exemplu, pentru rotație se specifică mai întâi componentele axei de rotație, apoi unghiul de rotație în radiani (în OpenGL se dă mai întâi valoarea unghiului în grade, apoi componentele axei de rotație).

Nodul `Transform` defineşte transformarea de coordonate a tuturor nodurilor fii ai acestuia față de nodul părinte. Prin aplicarea acestei transformări, un nod fiu este transformat din sistemul de referință propriu în sistemul de referință al nodului părinte. Succesiunea de transformări care se aplică unui nod frunză îl transformă din sistemul de referință local în sistemul de referință universal.

■ Exemplul 11.7

Fişierul `cupola.vrl` de mai jos construieşte o cupolă din două obiecte poziționate folosind un nod `Transform`:

```
#VRML V2.0 utf8
# cupola.wrl - construieşte o cupolă
Group {
    children [
        Background {
            skyColor  1.0 1.0 1.0
        },
        DirectionalLight {
            direction  0.0 -1.0 -1.0
            intensity 1.0
            ambientIntensity 0.2
            color 1.0 1.0 1.0
        },

        # Pereții cupolei
        Shape {
        appearance DEF Brown Appearance {
            material Material {
                diffuseColor 0.6 0.4 0.0
            }
        }
         geometry Cylinder {
                height 2.0
                radius 2.0
            }
    },
    # Acoperişul cupolei
    Transform {
            translation 0.0 2.0 0.0
            children [
                Shape {
                    appearance USE Brown
                    geometry Cone {
```

```
                        height 2.0
                        bottomRadius 2.5
                    }
                }
            ]
        }
```

Cupola este reprezentată printr-un nod `Group` cu doi fii. Primul fiu este un cilindru care redă pereții cupolei; cel de-al doilea fiu este un con translatat cu 2 unități față de nodul părinte (nodul `Group`) şi reprezintă acoperişul. Imaginea obținută prin încărcarea fişierului `cupola.wrl` este dată în fig. 11.11(a).

(a)

(b)

Fig. 11.11 (a) Imaginea fişierului cupola.wrl.
(b) Imaginea fişierului asterix.wrl

■ Exemplul 11.8

```
#VRML V2.0 utf8
# asterix.wrl - construieşte semnul "asterix" 3D
Group {
    children [
        Viewpoint {
            position 40.0, 10.0, 40.0
            orientation 0.0 1.0 0.0 0.78
        },
# Primul braţ
    DEF Arm1 Shape {
        appearance Appearance {
            material Material { }
        }
        geometry Cylinder {
            height 1.0
            radius 0.1
        }
    },
# Al doilea braţ
    Transform {
        rotation 1.0 0.0 0.0  1.047
```

```
            children USE Arm1
        },
    # Al treilea braț
        Transform {
            rotation 1.0 0.0 0.0  2.094
            children USE Arm1
        }
    ]
}
```

Fişierul `asterix.wrl` construieşte un semn asterix în trei dimensiuni. Obiectul este reprezentat printr-un nod `Group` cu patru fii. Primul nod fiu (`Viewpoint`) poziționează punctul şi direcția de observare. Cel de-al doilea nod fiu este un braț reprezentat printr-un cilindru vertical; cel de-al treilea fiu al nodului `Group` reprezintă al doilea braț şi se obține printr-o rotație cu 60° (1.047 radiani) a primului braț relativ la axa x; cel de-al patrulea fiu al nodului `Group` reprezintă al treilia braț şi se obține printr-o rotație cu 120° (2.094 radiani) a primului braț relativ la axa x. Grafurile scenelor corespunzătoare fişierelor `cupola.wrl` şi `asterix.wrl` sunt date în fig. 11.12.

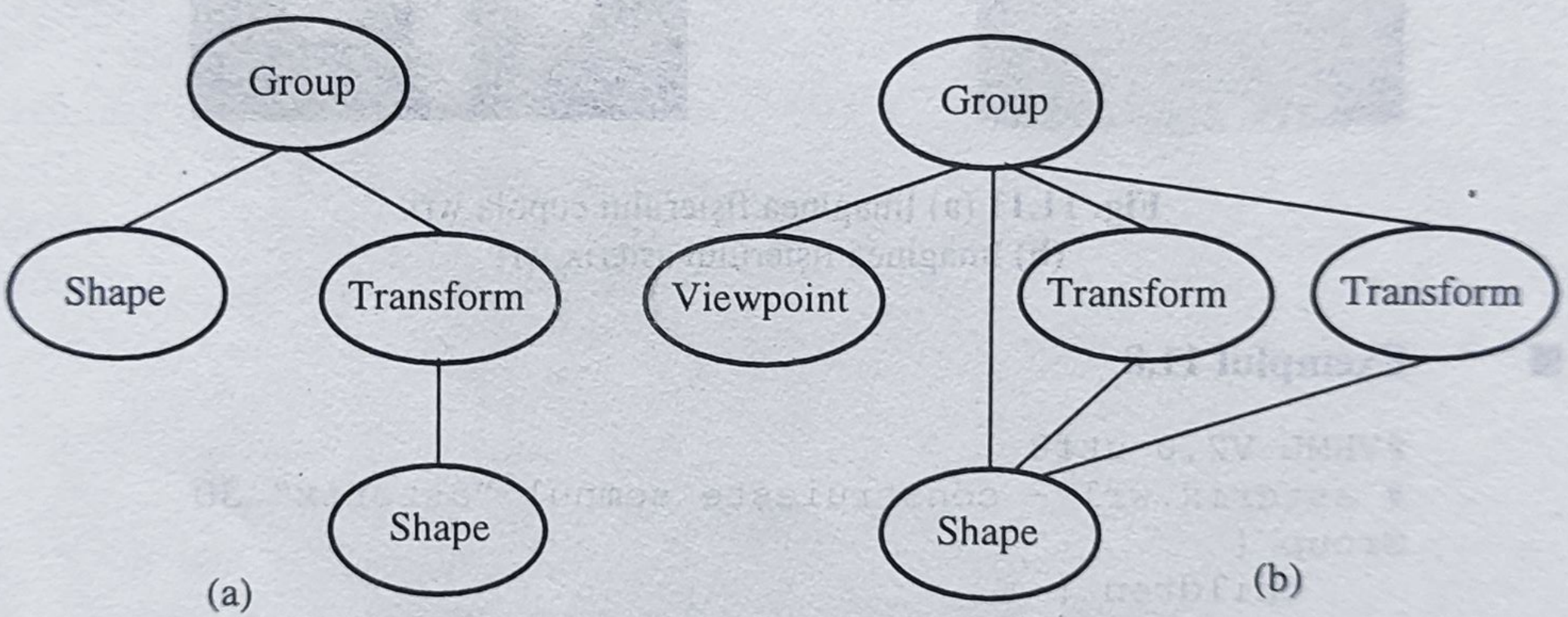


Fig. 11.12 (a) Graful scenei cupola.wrl (b) Graful scenei asterix.wrl.

Nodul `Switch` conține o listă ordonată de noduri fii care descriu reprezentări variate ale aceluiaşi obiect. În cursul redării, se selectează în secvență câte unul din fii la fiecare cadru de imagine, ceea ce creează impresia de evoluție (modificare dinamică) a obiectului. În acest fel sunt redate luminile de semnalizare (care îşi schimbă culoarea sau intensitatea la intervale stabilite de timp) sau flăcările, fumul, etc.

Nodul `LOD` conține reprezentarea unui obiect cu nivele de detaliu multiple. Fiecare nod fiu al nodului `LOD` este selectat într-un interval de distanțe specificat în vectorul de distanțe al nodului `LOD`.

Nodul `Billboard` defineşte o transformare a nodurilor fii, dar automat adaugă la această transformare o rotaţie astfel încât axa z a sistemului de referinţă al nodurilor fii să fie în permanenţă orientată către punctul de observare. În acest mod, atunci când observatorul se mişcă în scenă, el vede întotdeauna aceeaşi faţă a obiectului. Nodul de tip Billboard se foloseşte pentra redarea copacilor în scenele virtuale. Un copac se reprezintă printr-o singură faţă cu o textură care reproduce imaginea copacului şi care se roteşte astfel încât observatorul vede faţa texturată şi nu muchiile acesteia.

Nodul `Inline` este un nod de grupare care permite introducerea unuia sau mai multor noduri fii dintr-un alt fişier VRML extern specificat prin codul URL. În acest fel, o scenă virtuală se poate compune din mai multe module (fişiere) existente.

11.3.3 ANIMAŢIA ÎN VRML

Una din cele mai importante caracteristici ale limbajului VRML este abilitatea de a descrie animaţia obiectelor în scena virtuală. În VRML se pot modifica multe caracteristici ale scenei virtuale, ca de exemplu:

- poziţia, orientarea şi dimensiunea obiectelor;
- culoarea, materialul, textura şi parametrii de aplicaţie a texturii pe feţele obiectelor;
- culoarea, poziţia şi direcţia luminilor.

Caracteristica animaţiei în VRML este aceea că fiecare nod al scenei este tratat ca un element cu intrări şi ieşiri, iar prin conectarea acestora se obţine un "circuit cablat" care poate fi utilizat ca un traseu de animaţie, de-a lungul căruia se pot deplasa date în scenă. În limbajul VRML, o conexiune între două noduri din scenă se numeşte traseu (*route*), o dată care se deplasează se numeşte eveniment (*event*), iar intrările şi ieşirile în noduri se numesc evenimente de intrare (*eventsIns*), respectiv, evenimente de ieşire (*eventsOuts*).

De exemplu, dacă un nod are un câmp numit `rotation`, atunci nodul respectiv are un eveniment de intrare numit `set_rotation` şi un eveniment de ieşire numit `changed_rotation`.

Pentru crearea unui traseu de animaţie, se definesc noduri de transformare şi noduri de interpolare, care asigură o interpolare liniară a valorilor de deplasare, rotaţie sau modificare a culorilor.

În VRML sunt definite şase tipuri de noduri de interpolare: `PositionInterpolator`, `OrientationInterpolator`, `ScalarInterpolator`, `CoordinateInterpolator`, `NormalInterpolator`, `ColorInterpolator`. Prin conectarea nodurilor de transformare şi de interpolare (folosind comanda ROUTE), se obţine o mare varietate de animaţie a obiectelor în scenă.

11.3.4 CARE ESTE VIITORUL LIMBAJULUI VRML?

În momentul de faţă (anul 2000) se pregăteşte o nouă versiune a limbajului VRML, numită, deocamdată, VRML NG şi, de asemenea, noi toolkit-uri de proiectare a scenelor tridimensionale. De exemplu, Silicon Graphics şi Microsoft au anunţat toolkit-ul Farenheit, iar Sun a dezvoltat toolkit-ul Java3D pentru crearea aplicaţiilor 3D folosind limbajul Java.

La fel ca în VRML, Java3D permite reprezentarea scenelor 3D complexe printr-un graf al scenei. Nodurile fii ale grafului sunt forme geometrice, materiale, lumini, sunete, etc, iar nodurile părinte sunt noduri de transformare sau de grupare. De fapt, pentru un cunoscător al limbajului VRML, terminologia din Java3D este foarte familiară. Cele mai multe noduri din Java3D au funcţionalităţi similare celor din VRML, dar, în plus, sunt prevăzute funcţionalităţi suplimentare, cum ar fi combinarea culorilor prin transparenţă, definirea modului de filtrare a texturilor, inclusiv filtrarea mip-map, efectul ceţii, etc.

Versiunea 1.1 a toolkit-ului Java3D este deja disponibilă (gratuit) atât pentru sisteme Sun Solaris, cât şi pentru sisteme Windows NT sau 95. Specificaţii Java3D se pot găsi la adresele http://java.sun.com/products/java-nedia/3D/, http://www.sun.com/deskop/java3d/ sau http://java3d.sdsc.edu.

Se poate remarca deci o tendinţă de unificare a reprezentării scenelor virtuale pentru diferite platforme de calcul, sisteme de operare, limbaje de programare şi toolkit-uri de dezvoltare pe baza specificaţiilor limbajului VRML.

11.4 APLICAŢII ALE REALITĂŢII VIRTUALE

Aplicaţii ale realităţii virtuale se întâlnesc în numeroase domenii de activitate, bazate pe diferite sisteme hardware/software de dezvoltare [Bry95]. Unul din cele mai importante domenii de activitate în care se folosesc sisteme de realitate virtuală este cel al aplicaţiilor militare şi aerospaţiale, în care astfel de sisteme (*simulatoarele de antrenament*) reduc considerabil atât costurile de antrenament, cât şi pericolul de exersare a manevrelor dificile. Un alt domeniu de activitate, cu un număr mai mic de aplicaţii decât cel din domeniul aerospaţial, dar cu un rol deosebit de important în evitarea şi reducerea pierderilor de vieţi omeneşti, este medicina, în special chirurgia. Alte domenii în care sistemele de realitate virtuală au un rol important sunt: educaţie, artă, afaceri, proiectare, robotică, divertisment, cercetare [Burd94], [Earn93], [Earn95], [Slat99]. În continuarea acestui capitol vor fi prezentate pe scurt o parte din cele mai interesante aplicaţii ale realităţii virtuale.

11.4.1 SIMULATOARE DE ANTRENAMENT

Simulatoarele de antrenament se construiesc pentru formarea deprinderilor de manevrare a unor vehicule: aerospaţiale, navale, rutiere, sau de cale ferată.

Dintre acestea, cea mai mare dezvoltare au cunoscut-o simulatoarele din industria aeronautică şi aerospaţială. Costul deosebit de mare al aparatelor de zbor, dotarea acestora cu echipamente complexe de navigare sau luptă aeriană, cerinţele de efectuare a unor manevre complicate, cu mare viteză şi precizie, au impus necesitatea pregătirii personalului de zbor înainte de a porni în misiune.

Un simulator de zbor este, de asemenea, o instalaţie complicată care utilizează o mare varietate de tehnologii şi echipamente (echipamente hidraulice, electronice, sisteme de calcul în timp real, proiectoare optice, generatoare de imagini 3D, etc.) şi este cel mai potrivit loc unde pilotul se poate acomoda cu cabina aparatului real [Vin93]. Acest lucru este posibil deoarece cabina simulatorului reproduce cu fidelitate un aparat specific (de exemplu: Airbus 320, Concorde, Boeing 767, Mig 21, etc.). Dintre avantajele oferite de antrenarea pe simulatoare se pot aminti:

- Securitatea completă a instruirii, pentru pilot şi aparat, atât în condiţii normale cât şi în condiţii dificile şi periculoase (defectări ale echipamentelor aeronavei, condiţii meteorologice grele, etc.).
- Economie de combustibil şi evitarea uzurii aparatelor de zbor.
- Evaluarea obiectivă a performanţelor echipajului prin analiza evenimentelor de zbor simulat.

Simulatoarele de antrenament sunt sisteme de realitate virtuală semi-imersive, în care se combină echipanente reale (elementele de comandă din cabină) cu imaginea mediului virtual, cu sunetului virtual şi cu senzaţia de mişcare generată artificial. Pilotul este ambarcat în scaunul fixat rigid de cabină, iar scenariile de antrenament îl obligă să poarte centură în fazele de decolare şi aterizare. Cea mai puternică senzaţie de imersiune în mediul virtual este asigurată prin imaginea vizuală, care trebuie să reproducă condiţiile de mediu cât mai realist posibil: aeroportul cu pista şi clădirile înconjurătoare, terenul cu reperele naturale sau construite (râuri, păduri, şosele, poduri, oraşe, etc.) [Ion98c]. Toate aceste date constitue baza de date grafice, generată off-line, care este încărcată în memoria generatorului de imagine şi traversată în timp real în cursul antrenamentului. Calitatea imaginii generate, ca realism, rezoluţie, complexitate şi frecvenţă de actualizare, reprezintă un factor important în instruirea corespunzătoare a piloţilor. Datorită cerinţelor de putere de calcul deosebit de ridicată, generatoarele de imagine vizuală au fost realizate, în general, ca sisteme de *procesare paralelă şi distribuită*, în diferite arhitecturi special proiectate sau folosind staţii grafice.

Scenariile de antrenament ale piloţilor conţin, de regulă, exersarea manevrelor normale (decolare, aterizare, zbor de croazieră), ca şi a manevrelor care se impun în situaţii de avarii ale aparatului. Răspunsul pilotului pentru fiecare situaţie specială este comparat cu cerinţele bine precizate pentru aparatul respectiv şi introduse ca parte componentă a modelului aparatului. Manevrele greşite sunt identificate şi penalizate corespunzător. Din fericire, în simulator, o manevră greşită poate fi reluată ori de câte ori este nevoie pentru deprinderea ei corectă, ceea ce în zborul real nu este posibil, orice manevră greşită putând conduce la o catastrofă. Succesul antrenării poate fi apreciat prin evaluarea îndemânării obţinute

şi prin capacitatea pilotului de a efectua un zbor real, după antrenarea pe un simulator de zbor.

Din punct de vedere funcțional, un simulator de zbor se compune din mai multe sisteme intercorelate: sistemul de modelare a aeronavei, sistemul de comandă a aparatelor de bord (avionica), sistemul vizual (generatorul de imagine), sistemul sonor (generatorul de sunet), sistemul de mişcare, sistemul de efort în comenzi. Fiecare sistem al simulatorului este compus dintr-un modul de comandă, interfețe specifice şi dispozitive de intrare-ieşire.

Din punct de vedere constructiv, un simulator de zbor este compus din două subansamble: subansamblul de pilotaj şi subansamblul de comandă şi control, interconectate printr-o magistrală de date.

Subansamblul de pilotaj este alcătuit din cabina de pilotaj, amplasată pe o platformă care îi asigură mişcările necesare. Cabina este echipată cu toate comenzile de zbor corespunzătoare comenzilor reale şi cu toate aparatele de bord indicatoare, cu aspect şi funcționare cât mai apropiată de cea a aparatelor reale. În fața cabinei sunt amplasate dispozitivele de redare a imaginii mediului virtual (monitoare sau ecran şi proiectoare), precum şi difuzoarele pentru generarea sunetului virtual.

Subansamblul de comandă şi control este compus din stațiile de calcul conectate în rețea locală, care comandă atât cabina de pilotaj cât şi un post al instructorului conținând aparate de urmărire a zborului. Aparatele de urmărire a zborului din postul instructor nu mai sunt, în mod necesar, identice ca aspect cu cele din cabina reală, şi sunt, în general, aparate virtuale, generate pe unul sau mai multe displayuri conectate la stațiile din rețea. În plus, sunt prevăzute numeroase facilități de urmărire a traiectului de zbor, a anvelopelor de zbor (valorile limită ale parametrilor) precum şi redarea imaginii mediului virtual care dublează imaginea generată în cabină.

Din punct de vedere arhitectural, un simulator de zbor este compus dintr-un calculator central (stație), care este conectat în rețea cu un număr de alte stații în care sunt amplasate modulele de comandă ale diferitelor sisteme ale simulatorului.

Sistemul central al unui simulator este *sistemul de modelare a aeronavei*, care este executat pe stația centrală a simulatorului (consola sistem). Datele de intrare ale modelului sunt comenzile date de pilot prin acționarea echipamentelor din cabină, preluate de interfețele specifice şi transmise prin magistrala de date. Pe baza acestor comenzi are loc integrarea (în timp real) a datelor dinamice ale aeronavei, ale motoarelor de propulsie, ale echipamentelor electrice, hidraulice, navigație, armament, etc. În fiecare pas de iterație, pe baza mărimilor de stare şi a mărimilor de intrare, se generează comenzile pentru toate celelalte sisteme funcționale ale simulatorului.

Celelalte sisteme ale unui simulator (sistemul de comandă a aparatelor de bord/avionică, sistemul de imagine, sistemul de sunet, sistemul de mişcare, sistemul de efort în comenzi) prelucrează comenzile primite de la sistemul de modelare şi prezintă datele rezultate în urma prelucrării în mod specific fiecărui sistem: indicațiile aparatelor de bord, imaginea scenei virtuale proiectate pe ecranul

din cabină, sunetul generat în difuzoare, mişcarea platformei pe care este amplasată cabina, senzația tactilă de efort la acționarea manetelor sau pedalelor).

Pilotul acționează în orice moment în funcție de aceste date. Se poate spune că un simulator este un sistem în buclă închisă, închiderea buclei fiind asigurată de însuşi pilotul (fig. 11.13).

Conexiunile reprezentate în fig. 11.13 între sistemele componente ale unui simulator sunt conexiuni logice. În realitate, toate stațiile de calcul sunt conectate în rețea locală printr-un segment Ethernet prin care se transferă mesaje punct-la-punct sau mesaje de difuziune (broadcast). Magistrala de date care conectează subansamblul cabină de subansamblul de comandă şi control foloseşte, în general, transmisia multiplexată a datelor.

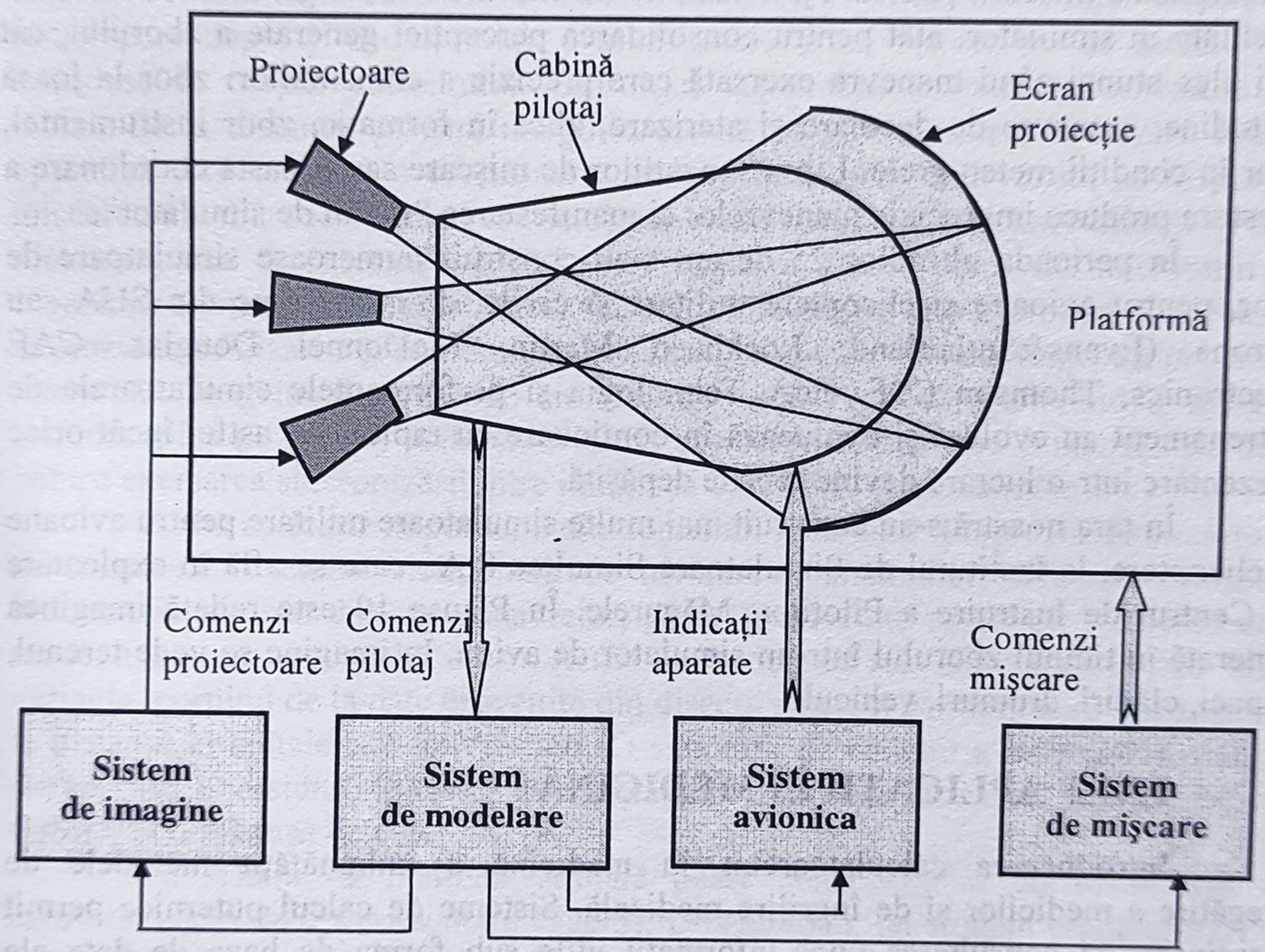


Fig. 11.12 Sistemele componente ale unui simulator de zbor.

Pentru o simulare de calitate sunt importante:

- Fidelitatea modelării.
- Fidelitatea redării zborului, aşa cum este perceput de către pilot (aparate de bord, scena vizuală, sunet, mişcare).
- Rată suficientă de mare de iterare a calculelor (pentru a crea impresia de continuitate).
- Timp de răspuns cât mai mic (intervalul de timp dintre momentul unei acțiuni a pilotului şi efectul acestei acțiuni, prezentat prin intermediul imaginii, indicațiilor aparatelor de zbor, etc.).

Una din senzațiile cele mai dificile de simulat este senzația de mişcare. În timpul zborului real, pilotul percepe mişcarea integrând informații provenind de la imaginea scenei exterioare (deplasarea orizontului şi a obiectelor în câmpul vizual), de la indicațiile aparatelor de bord (indicații de altitudine, viteză, orientare), senzații tactile (percepute de senzorii musculari), senzații de mişcare (percepute de sistemul vestibular al urechii).

Într-un simulator de zbor, senzația de mişcare se generează prin reprezentarea corespunzătoare a imaginilor (imaginea în mişcare a mediului virtual şi indicațiile aparatelor de bord) şi prin accelerarea cabinei simulatorului folosind platforme (în general cu 6 grade de libertate) acționate de sisteme hidraulice puternice. Cele două surse de senzații se completează reciproc, creând o percepție consolidată a mişcării [Botea97]. Senzațiile de mişcare trebuie generate cu maximă fidelitate în simulator, atât pentru consolidarea percepției generale a zborului, cât mai ales atunci când manevra exersată cere precizie a comenziilor: zbor la joasă altitudine, manevre de decolare şi aterizare, zbor în formație, zbor instrumental, zbor în condiții meteo grele. Lipsa senzațiilor de mişcare sau proasta coordonare a acestora produce imprecizia manevrelor şi manifestarea "răului de simulator".

În perioada ultimilor 25 de ani s-au construit numeroase simulatoare de zbor, pentru avioane şi elicoptere militare şi civile, de mari firme din SUA sau Europa (Evens&Sutherland, Lockheed Martin, McDonnel Douglas, CAE Electronics, Thomson CSF, etc.). Tehnologia şi performanțele simulatoarele de antrenament au evoluat şi evoluează în continuare cu rapiditate, astfel încât orice prezentare într-o lucrare devine repede depăşită.

În țara noastră s-au construit mai multe simulatoare militare pentru avioane şi elicoptere, la Institutul de Simulatoare Simultec S.A., care se află în exploatare în Centrul de Instruire a Piloților, Măgurele. În Planşa 10 este redată imaginea generată în timpul zborului într-un simulator de avion. În imagine se vede terenul, copaci, clădiri, drumuri, vehicule.

11.4.2 APLICAȚII ÎN MEDICINĂ

Introducerea calculatoarelor în medicină a îmbunătățit metodele de pregătire a medicilor şi de îngrijire medicală. Sisteme de calcul puternice permit menținerea şi consultarea unor informații utile sub forma de baze de date ale pacienților sau ale diagnosticelor, consultații la distanță, radiografii digitale, simulări prechirurgicale.

Utilizarea realității virtuale în chirurgie are un impact puternic în ceea ce priveşte antrenarea chirurgilor în practicarea unor noi proceduri chirurgicale, în planificarea operațiilor complexe sau în prezentarea informațiilor de "navigare" în cursul unor intervenții chiriurgicale.

Formarea unui chirurg cu adevărat competent în operații complicate necesită efectuarea unui număr foarte mare de operații, în mulți ani după absolvirea studiilor. Exersarea operației pe cadavre este foarte dificilă deoarece, după lezarea organului investigat, operația nu mai poate fi repetată. Posibilitatea ca un chirurg să repete o procedură chirurgicală, până la perfecțiune, înainte de a o efectua asupra

pacientului, permite scăderea riscului la care este supus pacientul. În sistemele de realitate virtuală de *simulare a operațiilor chirurgicale*, este importantă generarea unei imagini cât mai realiste a organelor investigate, precum şi simularea senzației tactile şi a forței de reacție, prin folosirea unei mănuşi de date.

Un astfel de simulator chirurgical pentru intervenții abdominale a fost dezvoltat de firma Division şi implementat pe două stații Silicon Graphics IRIS 310 VXG, care afişează imaginea steroscopică pe display-urile unei căşti de vizualizare HMD EyePhone. Sistemul este prevăzut de asemenea cu două mănuşi de date DataGlove [Sat92].

Una din cele mai delicate tehnici chirurgicale, *tehnica cu invaziune minimală* (*minimally invasive surgery*), necesită o antrenare intensă a chirurgului, datorită condițiilor în care acesta trebuie să opereze. În astfel de intervenții (cum sunt laparoscopia şi endoscopia), o minicameră şi un instrument de incizie sunt introduse în corpul pacientului. Chirurgul acționează asupra instrumentului de incizie de la distanță, privind imaginea redată pe un monitor TV. Pentru reuşita operației, este necesară o coordonare perfectă între imaginea văzută pe display şi mişcarea mâinii, coordonare care se obține numai printr-o antrenare intensă şi care nici nu poate fi exersată pe cadavre. Numărul foarte mare de intervenții cu invaziune minimală (apreciat la mai multe milioane pe an) a produs o adevărată explozie în realizarea şi comercializarea *simulatoarelor de intervenții cu invaziune minimală*. În general, astfel de simulatoare sunt imersive, generând o imagine stereoscopică pe display-urile unui HMD şi posedă obligatoriu o mănuşă de date, pentru exersarea sincronizării între imaginea văzută şi mişcarea mâinii.

O altă aplicație importantă din domeniul medicinii îl reprezintă *simulatorul de anatomie*. Metoda tradițională de învățare a anatomiei, folosind fotografii bidimensionale sau disecția cadavrelor, poate fi înlocuită astăzi cu studierea unui model anatomic al corpului uman. Astfel de modele au fost dezvoltate în mai multe variante, pornind de la date provenite din disecții şi pot fi achiziționate sau accesate la distanță în spitale sau universități. O simulare de calitate a *corpului virtual* se obține prin imersiune, folosind un sistem de vizualizare cu cască montată pe cap (HMD) şi o mănuşă de date.

Un sistem de realitate virtuală poate constitui atât un instrument de instruire, cât şi de experimentare [Sat95]. De exemplu, un student poate urmări un traseu într-un anumit organ sau sistem anatomic, după care poate experimenta o anumită intervenție în acesta. Astfel de sisteme se pot considera sisteme educaționale 4D: la spațiul tridimensional se adaugă a patra dimensiune, timpul; studentul "merge" prin stomac şi "vede" un ulcer, ceea ce permite o acumulare de cunoştințe incomparabilă cu aceea obținută din planşe sau disecții.

În medicină se mai întâlnesc numeroase alte aplicații ale realității virtuale: în telechirurgie, diagnosticare, recuperare, terapie [Nor98], [Lear97]. De exemplu, sisteme de realitate virtuale pot fi utilizate pentru tratarea fricii, a fobiilor sau a stresului, dar această utilizare poate fi riscantă şi este supusă unor reglementări medicale internaționale.

11.4.3 APLICAȚII ÎN ARTĂ ȘI DIVERTISMENT

Divertismentul reprezintă cea mai mare piață pentru produsele de realitate virtuală şi numeroase firme cu mare pondere financiară s-au implicat în astfel de aplicații. Există o mare varietate de forme de divertisment în medii virtuale, de la jocuri video la domiciliu, până la sisteme complexe cu imersiune.

De asemenea, au fost organizate numeroase expoziții de jocuri în realitate virtuală, în care se desfăşoară competiții între echipe de joc, prin intermediul unei rețele de comunicație care asigură o simulare interactivă distribuită. Caracteristica importantă a unor astfel de manifestări (cum au fost cele cu simulatoarele *Battle Tech* sau *Mirage*) este aceea că, deşi este vorba de divertisment, nivelul tehnic este deosebit de ridicat, atingând nivelul simulatoarelor militare şi asigurând o puternică senzație de realism. Astfel de expoziții au atras numeroşi vizitatori şi, bineînțeles, venituri importante.

Pe aceleaşi principii au fost create numeroase filme de science-fiction şi alte producții artistice, realitatea virtuală reprezentând un nou mijloc de expresie artistică. Tendința care se evidențiază în producțiile artistice prin intermediul realității virtuale este aceea de a crea o interacțiune spectator-calculator-artist sau chiar direct între spectatori, ceea ce dă produselor artistice (picturi, compoziții muzicale, piese de teatru) un caracter dinamic, în continuă schimbare. Această caracteristică diferențiază puternic arta virtuală de arta clasică, aşa cum este prezentată în muzee sau expoziții.

O idee fascinantă de utilizare a realității virtuale în artă o constitue *muzeele virtuale*. Orice posesor al unui calculator ar putea "vizita" oricare muzeu de pe glob, fără să se deplaseze de la propriul domiciliu. Bineînțeles, vizitarea muzeului virtual nu poate înlocui vizitarea muzeului real, decât în măsura în care sistemul grafic pe care se execută vizionarea este deosebit de performant.

Un alt experiment interesant îl constitue *teatrul virtual*. În anul 1998 un grup de cercetători canadieni au inițiat un proiect ambițios: montarea piesei lui Shakespereare, *Visul unei nopți de vară*, şi difuzarea acesteia pe Internet (sub numele de VRML Dream) [Mat99]. Decorul şi actorii au fost modelați în VRML 2.0, iar vocile au fost oferite de un grup de actori (reali), digitizate şi comprimate. În mai puțin de un an, VRML Dream a cunoscut o mare audiență pe Internet, unde poate fi accesat printr-un browser (ca, de exemplu, CosmoPlayer) la adresa http://www.vrmldream.com/vrmldream/artistic.html.

11.4.4 FACTORUL UMAN, ETIC ŞI ESTETIC ÎN REALITATEA VIRTUALĂ

Realitatea virtuală a devenit binecunoscută şi mediatizată în ultimii ani, dar, de multe ori, speculațiile au creat mituri despre ceea ce este sau ce ar putea realiza aceasta, mituri care nu corespund întotdeauna cu starea de fapt. Realitatea virtuală a creat o nouă formă de interacțiune om-calculator, cu o interfață fundamental nouă. În locul utilizării în mod convențional a tastaturii şi ecranului,

utilizatorul poartă o cască de vizualizare pe cap, iar o mănuşă de date permite interacțiunea cu mediul virtual.

Esența realității virtuale este imersiunea, participanții având senzația că fac parte din mediul virtual. Sistemele de realitate virtuală se află într-o evoluție extrem de rapidă, în care dezvoltările hardware şi software au loc în mod permanent şi sunt accesibile pentru un număr din ce în ce mai mare de utilizatori. În consecință, studierea realității virtuale din punct de vedere al factorului uman, etic şi estetic, precum şi al posibilelor efecte colaterale asupra utilizatorilor, a devenit o necesitate.

Factorul uman. "Răul de mişcare" (*motion sickness*) este binecunoscut şi documentat. Este vorba atât de "răul de mare", resimțit în călătoriile cu vaporul, cât şi de "răul de zbor", resimțit în aeronave sau nave spațiale. O altă formă de "rău de mişcare" este "răul de simulator", raportat de utilizatorii simulatoarelor de antrenament încă din anul 1957. Acesta constă din numeroase modificări funcționale la nivel gastric, hormonal, cardiovascular şi respirator, cu manifestări de amețeală, greață, paloare, tremurat, etc.

Numeroase studii efectuate asupra "răului de simulator", pe mii de simulatoare şi piloți antrenați, au stabilit că, în majoritatea cazurilor, în timp, are loc o acomodare şi o scădere a simptomelor adverse. Există însă şi cazuri în care acomodarea poate să nu aibă loc niciodată. De asemenea, se cunosc şi simptome adverse care persistă, mai mult sau mai puțin timp, după participarea într-un simulator. Considerându-se că "răul de simulator" este o problemă serioasă, care poate reduce puternic eficiența antrenării, s-au dezvoltat mai multe teorii asupra cauzelor care îl provoacă. Majoritatea acestora au stabilit că simptomele adverse sunt cauzate de un *conflict senzorial* şi apar datorită neconcordanței între stimulii vizuali şi cei vestibulari sau între stimulii primiți şi ceea ce persoana se aşteaptă să simtă, conform experiențelor anterioare [Reg95].

Realitatea virtuală imersivă implică un conflict senzorial datorită neconcordanței între stimulii vizuali şi stimulii vestibulari. În cele mai multe sisteme de realitate virtuală imersive, mişcarea mâinii (echipată cu un dispozitiv de interacțiune) este cea care provoacă modificarea imaginii, în timp ce poziția corpului (care controlează sistemul vestibular) rămîne neschimbată. La acest conflict se mai adaugă conflictul datorat întârzierii de răspuns a sistemului, ceea ce provoacă neconcordanță de timp între mişcarea fizică reală, care acționează ca stimuli vestibulari, şi modificarea imaginii, care acționează ca stimuli vizuali.

Rezultate experimentale, efectuate pe numeroase sisteme de realitate virtuală şi numeroase echipe de utilizatori [Ken89], au confirmat existența simptomelor adverse şi amenințarea pe care acestea o reprezintă pentru extinderea utilizării sistemelor de realitate virtuală. De aceea, studierea în continuare a cauzelor "răului de simulator", pentru a putea fi eliminat sau redus, este de o mare importanță pentru viitorul sistemelor de realitate virtuală.

Factorul etic. Experimentele de terapie psihiatrică prin intermediul realității virtuale trebuie privite cu circumspecție, cu atât mai mult cu cât progresul

tehnologic le fac accesibile unui număr mare de experimentatori entuziaşti, dar lipsiți de cunoştințe ştiințifice temeinice.

De aceea, în privința utilizării realității virtuale în recuperare şi terapie, s-a conturat necesitatea unor reglementări de etică medicală, care să împiedice abuzul sau experimentele pseudo-ştiințifice, cu posibile efecte nocive asupra pacienților [Wha93]. Astfel de reglementări au apărut deja, atât în SUA cât şi în Europa, şi ele statuează principiile unei *practici clinice corecte* (*Good Clinical Practice*, 1990). Aceste principii includ obligativitatea obținerii acordului subiectului de a fi supus cercetărilor sau experimentelor medicale (inclusiv a celor bazate pe sisteme de realitate virtuală).

Factorul estetic. Rolul educativ al multor expoziții sau muzee virtuale este unanim recunoscut, dar, de multe ori, experimentele oferite sunt lipsite de profunzime estetică sau culturală. De aceea, se încearcă implicarea artiştilor contemporani în crearea galeriilor de artă virtuală, pentru a asigura nivelul estetic al artei construite prin intermediul realității virtuale.

O astfel de implicare a fost încercată în anul 1994, când a fost lansat proiectul *Galeria Virtual* [Par95], la care au participat numeroşi artişti, invitați să-şi exprime propria viziune asupra unei "realități" care extinde realitatea cunoscută. Rezultatele obținute în *Galeria Virtual*, ca şi în alte manifestări de acest gen, conturează factorul creativ şi estetic al producțiilor artistice bazate pe realitatea virtuală.

BIBLIOGRAFIE

[Adam93] **J. Adam**, "*Virtual Reality is for Real*", IEEE Spectrum, IEEE Press, October, 1993.

[Ames97] **A.L. Ames, D.R. Nadeau,** şi **J.L. Moreland**, *The VRML 2.0 Sourcebook*, John Wiley & Sons Inc., New York, 1997.

[Aur91] **F. Aurenhammer**, "*Voronoi Diagrams-A Survey of Fundamental Geometric Data Structure*", ACM Computing Survey, Vol. 23 Nr. 3, Sept. 1991.

[Azu97] **R.T Azuma**, "*A survey of Augnented Reality*", Presence: Teleoperators and Virtual Environments, Vol. 6, Nr. 4, August 1997, pp. 355-385.

[Baciu99] **Rodica Baciu** şi **D. Volovici**, *Sisteme de prelucrare grafică*, Editura Albastră, Cluj-Napoca, 1999.

[Bez72] **P. Bezier,** "*Numerical Control: Mathematics and Applications*", Chichester-Wiley, 1972.

[Bla98] **E. Blake**, *Interactive Computer Graphics*, online, http://www.cs.utc.ac.za/Courses/CS300WICG/

[Bier86] **E.A.Bier** şi **K.R. Sloan**, "*Two-part texture mapping*", IEEE Computer Graphics and Applications, Vol. 6, Nr. 9, 1986, pp. 40-5.

[Botea97] **C. Botea**, *Comanda mişcării simulatoarelor de zbor*, teză de doctorat, Universitatea Polithnica Bucureşti, 1997.

[Bre65] **J.E. Bresenham,** "*Algorithm for computer control of digital plotter*", IBM Systems Journal, January 1965, pp. 25-30.

[Brown95] **D.J. Brown**, **S.V. Cobb** şi **R.M.Eastgate**, "*Learning in virtual environments*", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1995.

[Bry95] **S. Bryson**, "*Approaches to the succesful design and implementation of VR applications*", R.A.Earnshow (ed), Virtual Reality Applications, Academic Press, London 1995.

[Bucur97] **C. Bucur** şi **Felicia Ionescu,** "*Debriefing Software as a Tool for Airline Operation Efficiency Improvement*", in Proceedings of the 8th International Training and Education Conference ITEC'97, pp. 199-203, April 1997, Lausanne, Switzerland.

[Burd92] **G. Burdea, J. Zhuang, E. Roskos, D. Silver** şi **N. Langrana**, "*A Portable Dextrous Master with Force Feedback*", Presence- Teleoperators and Virtual Environments, Vol. 1, Nr. 1, pp 18/27, March 1992.

[Burd93] **G. Burdea**, "*Virtual Reality Systems and Applications*", Electro'93 International Conference, Short Course, Edison, NJ, April 1993.

[Burd94] **G. Burdea** şi **P. Coiffet**, *La Réalité Virtuelle*, Edition Hermes, Paris, 1994.

[Burd97] **G. Burdea**, *Force and Touch Feedback for Virtual Reality*, John Willey & Sons, Inc., New York, 1997.

[Car97] **M.S.T. Carpendale, D.J. Cowperthwaite** şi **D. Fracchia**, "*Extended Distorsion Viewing from 2D to 3D*", IEEE Computer Graphics and Applications, Vol. 17, No. 4, 1997.

[Car84] **L.C. Carpenter**, "*The A-buffer, an anti-aliasing hidden surface method*", Computer Graphics, Vol. 18, Nr. 3, 1984, pp. 103-108.

[Car99] **G.S. Carson, R.F. Puk** şi **R. Carey**, "*Developing the VRML 97 International Standard*", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.

[Cat75] **E. Catmull**, "*Computer display of curved surfaces*", Proc. IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures, May 1975.

[Cat78] **E. Catmull**, "A hidden surface algorithm with anti-aliasing", Computer Graphics, Vol. 12. Nr. 3, 1978, pp. 6-10.

[Chau97] **J.C. Chauvin**, "*Geographical Coordinates in Real-Time Image Generation*", Proceedings of the 8th International Training and Education Conference ITEC'97, April 1997, Lausanne, Switzerland.

[Cohen97] **D. Cohen-Or** şi **A. Kaufman**, "*3D Line Voxelization and Connectivity Control*", IEEE Computer Graphics and Applications, Vol. 17, No. 6, Nov/Dec. 1997.

[Craw81] **F.C. Craw**, "*A comparison of anti-aliasing techniques*" IEEE Computer Graphics and Applications, Vol. 1, Nr. 1, pp. 40-48, 1981.

[Cig97] **P. Cignoni**, **E. Puppo** şi **R. Scopigno**, "*Multiresolution Representation and Visualization of Volume Data*", IEEE Transaction on Visualization and Computer Graphics, Vol. 3, No. 4, October-December 1997.

[Del97] **C. Delepine**, "*Online terrain level of detail*", Proceedings of the 8th International Training and Education Conference ITEC'97, pp. 23-28, April 1997, Lausanne, Switzerland.

[Dev96] **V. Devarajan**, **R.Fuentes** şi **D.E. McArthur**, "*An Approach to multiple levels of detail generation from Digital Terrain Elevation Data using Wavelet Transforms*", Proceedings of the 7th International Training Equipment Conference ITEC'96, pp. 255-262, April 1996, The Hague, The Netherlands.

[Dog88] **D. Dogaru**, *Elemente de grafică 3D*, Ed. Ştiinţifică şi Enciclopedică, Bucureşti, 1988.

[Drag57] **M. Drăganu**, *Introducere matematică în fizica teoretică modernă*, Editura Tehnică, 1957.

[Fang95] **Tsung-Pao Fang** şi **Les A. Piegl**, "*Delaunay Triangulation in Three Dimensions*", IEEE Computer Graphics and Applications, Vol. 15, Nr. 5, September 1995, pp. 62-69.

[Earn93] **R.A. Earnshaw, M.A. Gigante** şi **H. Jones**, *Virtual Reality Systems*, Academic Press, London, 1993.

[Earn95] **R.A. Earnshaw, J.A. Vince** şi **H. Jones**, *Virtual Reality Applications*, Academic Press, London, 1995.

[Earn97] **R.A.Earnshaw**, "*3D Multimedia on the Information Superhighway*", IEEE Computer Graphics and Applications, Vol. 17, No. 2, March/April 1997.

[Fjall93] **Per-Olof Fjallstrom**, "*Evaluation of a Delaunay-based method for surface approximation*", Computer-Aided Design, Vol. 25, Nr. 11, November 1993, pp. 711-719.

[Foley90] **J.D. Foley** şi **A. Van Dam**, *Computer Graphics: Principles and Practices*, 2nd Edn, Addison-Wesley, Reading, Massachusetts, 1990.

[Ger97] **N. Gershon** şi **S.G. Eick**, "*Information Visualization*", IEEE Computer Graphics and Applications, Vol. 17, No. 4, July/August 1997.

[Gig93] **M.A. Gigante**, "*Virtual Reality: Definitions, History and Applications*", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993.

[Gold94] **B. Goldiez** şi **Karen Williams**, "*Developing Simulator Netwoks*", Proceedings of the International TrainingEquipment Conference ITEC'94, April 1994, The Hague, The Netherlands.

[Gour71] **H. Gouraud,** "*Continous Shading of Curved Surfaces*", IEEE Trans. on Computers, C-20(6), June 1971, pp. 623-629.

[Grim93] **M.A. Grimsdale,** "*SuperVision – A Parallel Architecture for Virtual Reality*", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993.

[Gross96] **M.H. Gross**, **O.G. Staadt** şi **R. Gatti**, "*Efficient Triangular Surface Approximation using Wavelets and Quadtree Data Structures*", IEEE Transaction on Visualization and Computer Graphics, Vol. 2 No. 2, June 1996.

[Hua75] **T.S. Huang**, *Picture Processing and Digital Filtering*, Springer-Verlag, Berlin, 1975.

[Iacob96] **A. Iacob, C. Bucur, C. Englert** şi **Felicia Ionescu**, "*Romanian Approach, Achievements and Experience in Airline Pilots' Low Cost Training*", in Proceedings of the East European Civil Aviation Training Conference, pp.73- 76, December 1996, Prague, Czech Republic.

[Iacob97] **A. Iacob** şi **Felicia Ionescu**, "*Dynamics Model vs Visual Database Interaction in a Tank Driver* Simulator", Proceedings of the 8th International Training and Education Conference ITEC'97, pp. 71-75, April 1997, Lausanne, Switzerland.

[Ion96a] **Felicia Ionescu**, "*Baze de date grafice orientate pe obiecte*", Revista Română de Informatică şi Automatică, vol.6 (1996), pp. 37-43.

[Ion96b] **Felicia Ionescu**, "*Partiţionarea Dinamică a Spaţiului de Traversare a Bazelor de Date Grafice*", Revista Română de Informatică şi Automatică, vol.6 (1996), pp. 67-73.

[Ion96c] **Felicia Ionescu**, "*Scalability of Real-Time Parallel Image Synthesis Systems*", Revue Roumaine des Sciences Techniques. - Électrotechnique et Énergétique, vol. 41 (1996), pp. 501-511.

[Ion96d] **Felicia Ionescu** şi **Ş. Alexa**, "*PC-Based Software System for Visual Data Base Creation*", Proceedings of the 7th International Training Equipment Conference ITEC'96, pp. 272-278, April 1996, The Hague, The Netherlands.

[Ion96f] **Felicia Ionescu**, "*Mapping Image Rendering Operations onto Parallel Processors*", in Proceedings of the International Semiconductor Conference CAS'96, pp. 167-170, October 1996, Sinaia, Romania.

[Ion96g] **Felicia Ionescu**, "*Deadlock-Free Partial Multinode Broadcast in Image Rendering Parallel Networks*", in Proceedings of the International Symposium Communications'96, pp. 147-152, November 1996, Bucharest, Romania.

[Ion97a] **Felicia Ionescu**, "*Controlul dinamic al granularităţii proceselor paralele de sinteză de imagine*", Revista Română de Informatică şi Automatică, vol. 7, (1997), pp. 5-11.

[Ion97b] **Felicia Ionescu,** "*Scheduling Real-Time Image Synthesis Operations in Multiprocessor Systems*", Studies in Informatics and Control, vol. 6 (1997), pp. 173-180.

[Ion97c] **Felicia Ionescu**, "*A Solution for Multiple Levels of Detail Terrain Generation*", Proceedings of the 8th International Training and Education Conference ITEC'97, pp. 23-28, April 1997, Lausanne, Switzerland.

[Ion97d] **Felicia Ionescu**, "*Codesign of a Parallel Architecture for Visual Systems*", in Proceedings of the 11th International Conference on Control Systems and Computer Science CSCS-11, pp. 242-245, May1997, Bucureşti, Romania.

[Ion98a] **Felicia Ionescu, C. Gheorghiu, C. Englert, E. Popa** şi **Iulia Suciu**, "*Optimization of the execution time in a distributed system implementing a full flight simulator*", in Proceedings of the 9th International Training and Education Conference ITEC'98, pp. 509-514, April 1998, Lausanne, Switzerland.

[Ion98b] **Felicia Ionescu, C. Gheorghiu, C. Englert** şi **C. Stratulat,** "*Communications Protocols in Distributed Synthetic Environments*", in Proceedings of the International Conference Communications'98, pp. 507-512, November1998, Bucharest, Romania.

[Ion98c] **Felicia Ionescu**, "*Modelarea multirezoluţie a formelor de teren*", Revista Română de Informatică şi Automatică, vol. 8, (1998), pp. 45 – 52.

[Ion99a] **Felicia Ionescu**, *Principiile Calculului Paralel*, Editura Tehnică, Bucureşti 1999.

[Ion99b] **Felicia Ionescu** şi **M. Ionescu**, "*Pipeline Processing of Shared Graphs in Multiprocessor Systems*", International Conference on Parallel and Distributed Processing Techniques and Applications, pp 1958-1963, June 1999, Las Vegas, Nevada, USA.

[Ion00a] **Felicia Ionescu**, "*Application-level Virtual Memory Management in Real-Time Multiprocessor Systems*", in Proceedings of the 14th Annual Symposium for Applied Computing ACM SAC-2000, March 2000, Como, Italy.

[Ion00b] **Felicia Ionescu** şi **A. Jalbă**, "*Data Parallelism vs. Control Parallelism in Synthetic Environments Visualization*", in Proceedings of the 11th International Training and Education Conference ITEC'2000, April 2000, The Hague, The Netherlands.

[Kal89] **Y.E. Kalay**, *Modeling Objects and Enviromnent*, Wiley&Sons, New York, 1989.

[Ken89] **R.S. Kennedy, M.G. Lilienthal, K.S. Berbaum** şi **M.E. McCauley**, "*Simulator sickness in U.S Navy Flight Simulators*", Aviation, Space and Environmental Medicine, 1989.

[Lear97] **Anne C. Lear**, "*Virtual Reality provides Real Therapy*", IEEE Computer Graphics and Applications, Vol. 17, No. 4, July-August 1997.

[Lou97] **M. Louka**, *An Introduction to Virtual Reality*, Ostfold Colledge, 1997.

[Man95] **C. Manetta şi R. Blade**, "*Glossary of Virtual Reality Terminology*", International Journal of Virtual Reality, Vol.1, Nr. 2, 1995.

[Mat99] **S.N. Matsuba şi B. Roehl**, "*The making of VRML Dream*", IEEE Computer Graphics and Applications, March/April, Vol. 19, Nr. 2, 1999.

[Mold96] **Florica Moldoveanu, Zoea Racoviţă, Ş. Petrescu, G. Hera** şi **M. Zaharia**, *Grafica pe Calculator*, Editura Teora, 1996.

[Nad99] **D.R. Nadeau**, "*Tutorial: Building Virtual Worlds with VRML*", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.

[New72] **M.E. Newell, R.G. Newell** şi **T.L. Sancha**, "*A new Approach to the shaded picture problem*", Proc. ACM National Conference, pp. 443-450, 1972.

[New81] **W.M. Newman** şi **R.F. Sproull**, *Principles of Interactive Computer Graphics*, McGraw-Hill, New Zork, 1981.

[Nor98] **M.M North, S.M. North**, şi **J.R. Coble**, "*Virtual Reality Therapy: An effective Treatment for the Fear of Public Speaking*", Int'l J. of Virtual Reality, Vol. 3, No. 2, 1998.

[Pap77] **A. Papoulis**, *Signal Analysis*, McGrow-Hill, New York, 1977.

[Par95] **R. Pares i Burgues** şi **N. Pares i Burgues**, "*GaleriaVirtual: a platform for non-limitation in comtemporary art?*", R.A.Earnshow (ed), Virtual Reality Applications, Academic Press, London 1995.

[Prep85] **F.P. Preparata** şi **M.I. Shamos**, *Computational Geometry: an Introduction*, Springer-Verlag, New-York, 1985.

[Phong75] **B. Phong**, "*Illumination for computer-generated pictures*", Comm. ACM, Vol. 18, Nr. 6, 1975, pp. 311-317.

[Reg95] **E.C. Regan**, "*Some human factors issues in immersive virtual reality: fact and speculation*", R.A.Earnshow (ed), Virtual Reality Applications, Academic Press, London 1995.

[Rob93] **W. Robunett** şi **J.P. Rolland**, "*A Computational Model for Stereoscopic Optics of a Head-Mounted Display*", R.A. Earnshaw (ed), Virtual Reality Systems, Acad. Press, London, 1993.

[Row93] **T.W. Rowley**, "*Virtual Reality Products*", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993.

[Sat92] **R. Satava,** "*Virtual Reality Surgical Simulator*", Proceeding Medicine Meets Virtual Reality Conference, San Diego, CA, 1993.

[Sch98] **M. Schulz**, **T. Reuding** şi **T. Ertl**, "*Analyzing Engineering Simulations in a Virtual Environment*", IEEE Computer Graphics and Applications, Vol. 18, No. 6, 1998.

[SGI96] **Silicon Graphics**, *Onyx2 Technical Report*, Mountain View, CA, 1996.

[Slat99] **M. Slater, D.P. Pertaub** şi **A. Steed,** "*Public Speaking in Virtual Reality*, IEEE Computer Graphics and Applications, March-April 1999.

[Suth74] **I.E. Sutherland** şi **G.W. Hodgman**, "Reentrant polygon clipping", Comm. ACM, Vol. 17, No. 1, pp. 32-42, 1974.

[Rour93] **Joseph O'Rourke**, *Computational Geometry in C*, Cambridge University Press, Boston, Massachusetts, 1993.

[Sat95] **R.M. Satava**, "*Virtual Reality for the phisicians of the 21st century*", R.A.Earnshow (ed), Virtual Reality Applications, Academic Press, London 1995.

[Sla99] **M. Slater, D.P. Pertaub** şi **A. Steed**, "*Public Speaking in Virtual Reality*", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.

[Sher92] **B. Sherman** şi **P Judkins**, *Glimpses of Heaven, Vision of Hell: Virtual Reality and its implications*, Hodder and Stoughton, London, 1992.

[Vin93] **J. Vince**, "*Virtual Reality Techniques in Flight Simulation*", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993

[Vin97] **J. Vince**, *Virtual Reality Systems*, Addison-Wsley, New York, 1997.

[Watt95] **A. Watt**, *3D Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1995.

[Wei77] **K. Weiler** şi **P. Atherton**, "*Hidden surface removal using polygon area sorting*", Computer Graphics, Vol. 11, No. 2, 1977, pp. 214-222.

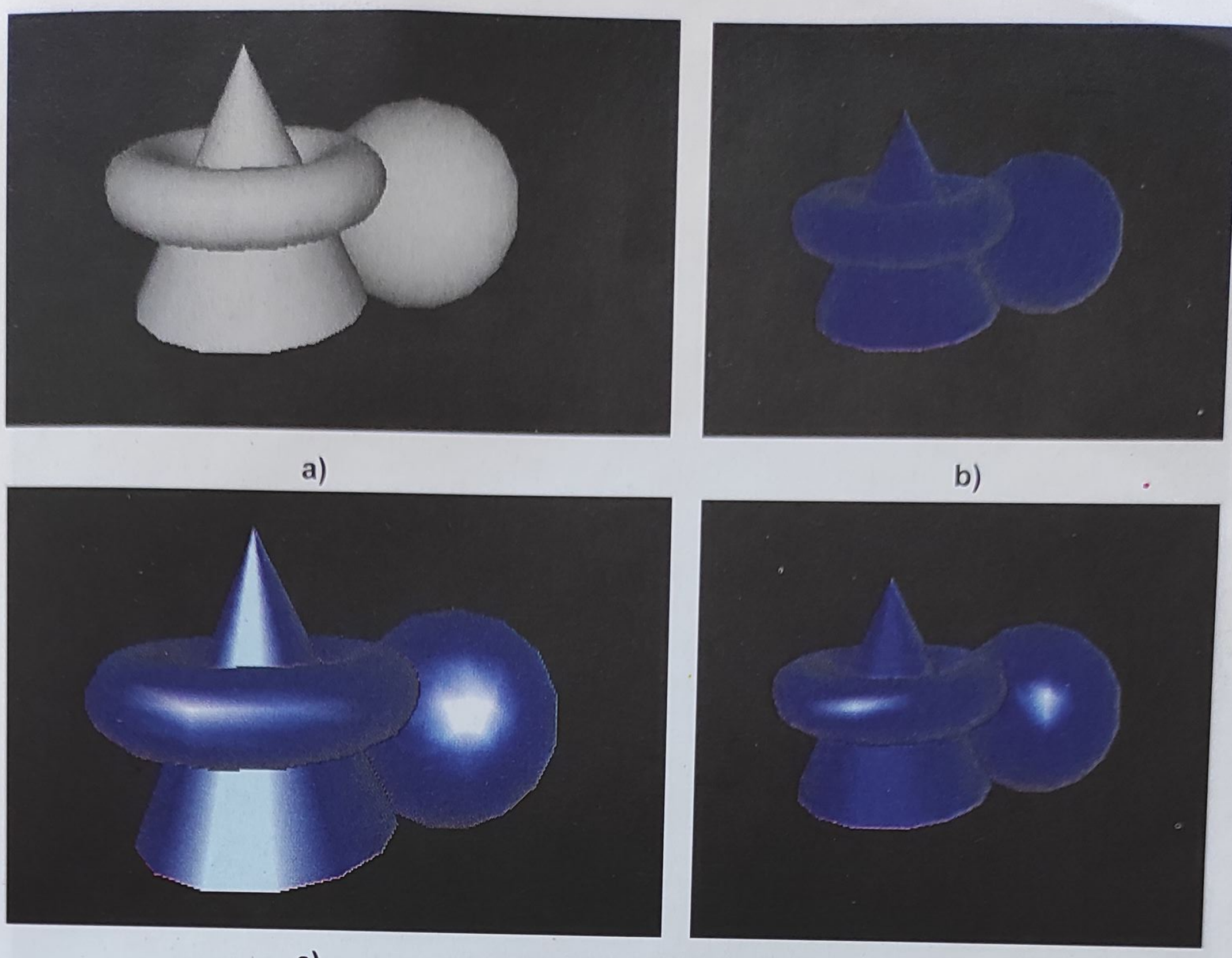
[Wha93] **L.J. Whalley**, "*Ethical Issues in VR Treatment of Mental Disorders*", R.A. Earnshow (ed), Virtual Reality Systems, Academic Press, London 1993.

[Woo97] **M. Woo, Jackie Neider** şi **T. Davis**, *OpenGL Programming Guide*, OpenGL Architecture Review Board, Addison-Wesley Developers Press, Reading, MA 1997.

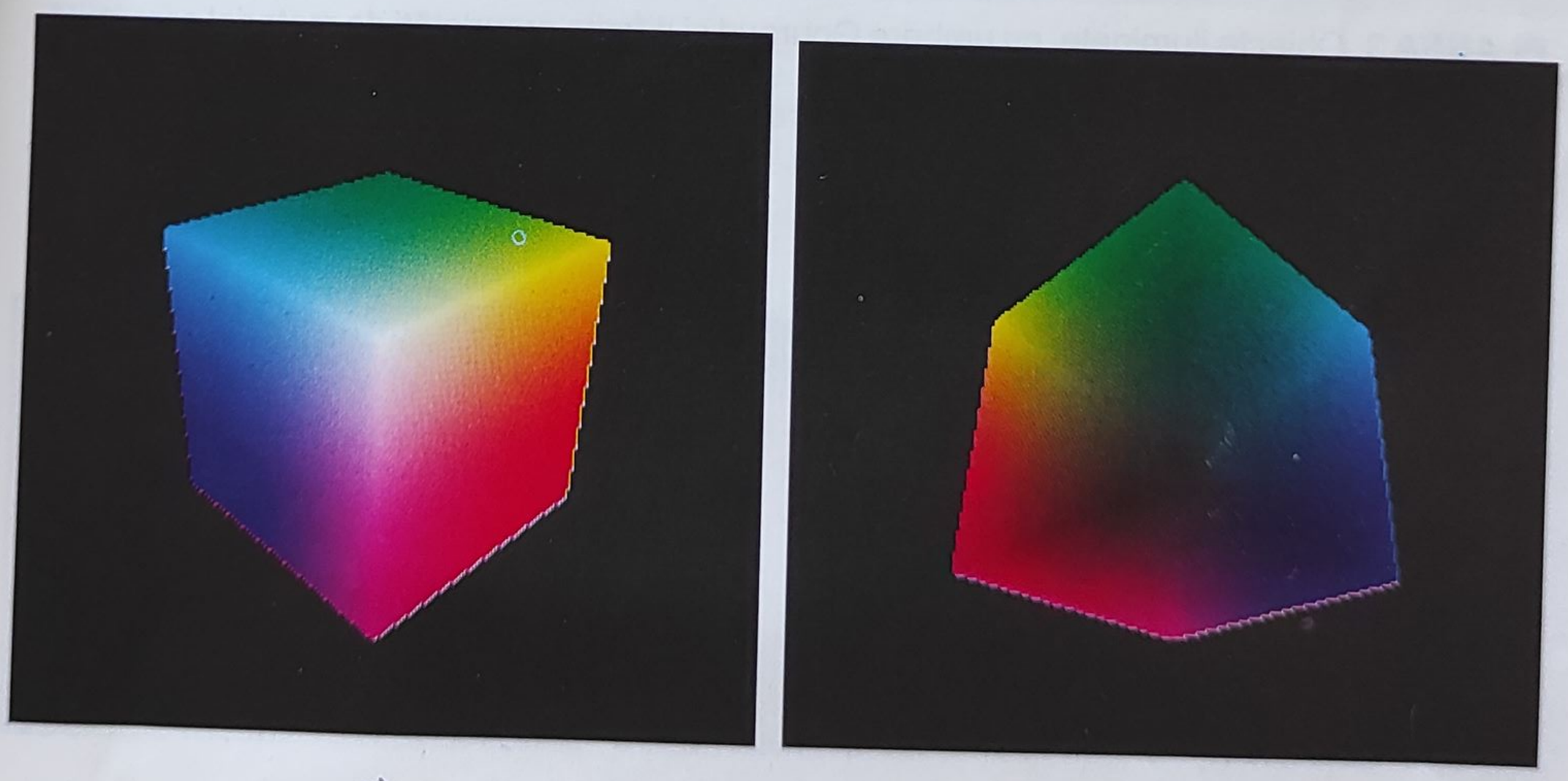
[Xia97] **Julie C. Xia, J. El-Sana** şi **A. Varshney**, "*Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models*", IEEE Transactions on Visualization and Graphics, Vol. 3, No. 2, April-June 1997.

[You98] **P. Young**, *3D Graphics and Virtual Reality*, online http://pc-vrg.dur.ac.uk/cg-notes/

INDEX BILINGV DE TERMENI



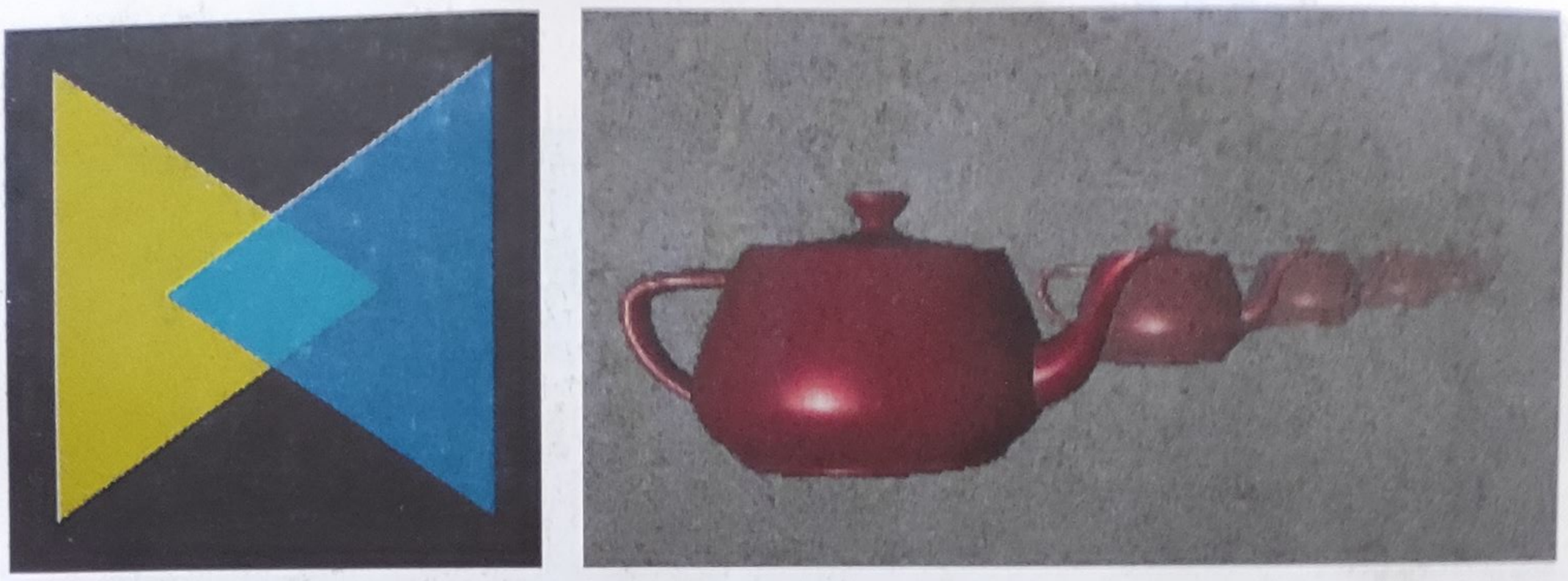
PLANŞA 1. Obiecte din material cu reflectanță ambientală, difuză şi specular gri, şi sursă de lumină colorată: **a)** obiectele iluminate de o sursă de lumină cu componente ambientală şi difuză gri; **b)** obiectele iluminate de o sursă de lumină cu componenta ambientală gri şi componenta difuză albastră; **c)** şi **d)** obiectele iluminate de o sursă de lumină cu componenta ambientală gri, componenta difuză albastră şi componenta specular albă; în **c)** exponentul de reflexie specular are valoare mică (20); în **d)** exponentul de reflexie specular are valoare mare (80).



PLANŞA 2. Cubul culorilor RGB: **a)** axele (Cyan, Magenta, Yellow - CMY); **b)** axele (Red, Green, Blue).

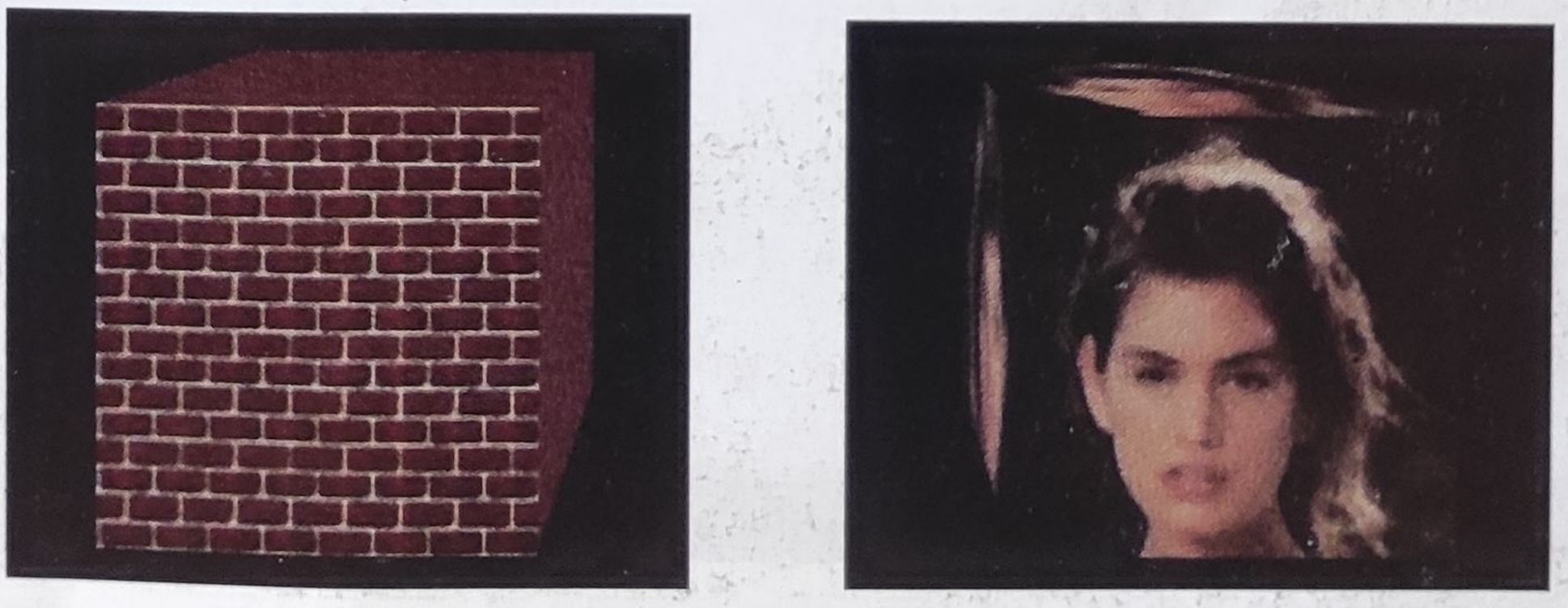
PLANŞA 3. Obiecte iluminate, cu umbrire Gouraud şi diferite proprietăți de material care imită materiale reale. *Pe prima coloană* (de sus în jos): jad, obsidian, perlă, rubin, turcoaz. *A doua coloană:* bronz, crom, cupru, aur, argint. În a treia coloană sunt obiecte din materiale plastice de diferite culori: cian, verde, roşu, alb, galben. În coloana a patra sunt obiecte din cauciuc cu culori similare.

PLANŞA 4. Obiecte redate cu iluminare şi umbrire. În partea stângă, sferă cu umbrire poligonală şi umbrire Gouraud. În partea dreaptă, suprafață Bézier cu umbrire Gouraud.



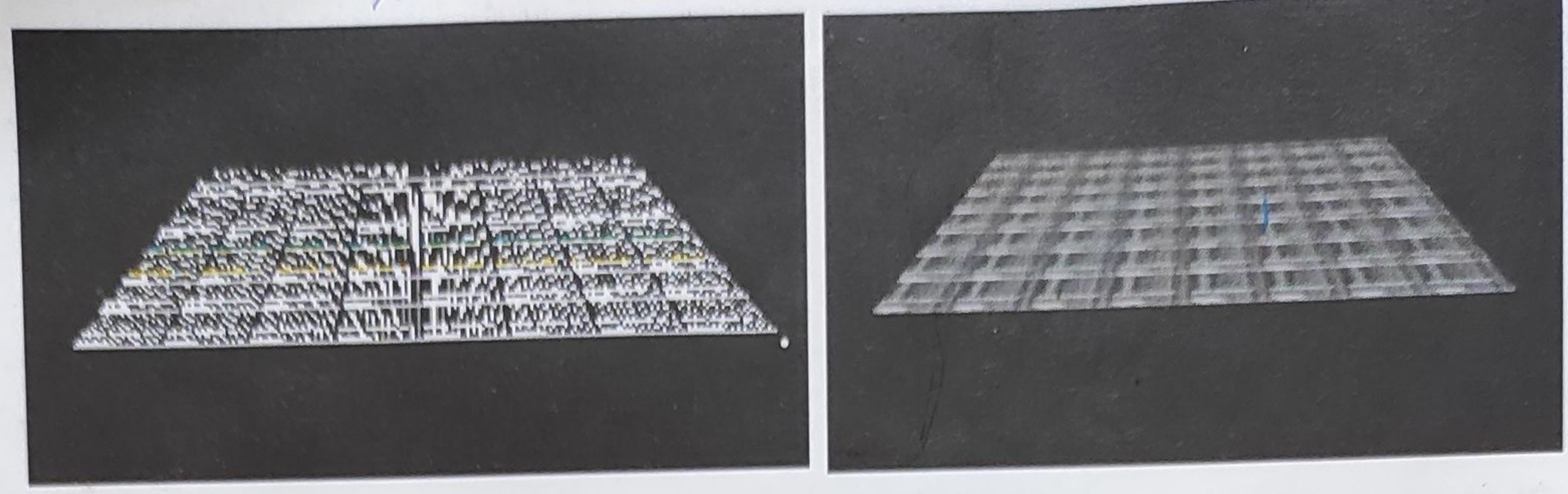
PLANŞA 5. Combinarea culorilor. În partea stângă, suprafețe transparente suprapuse; triunghiul de culoare cian este transparent şi suprapus peste triunghiul de culoare galbenă. În partea dreaptă, simularea efectelor atmosferice (ceață).

PLANŞA 6. Obiecte cu iluminare şi umbrire. La stânga, obiectul este desenat cu aliasing. La dreapta, obiectul este desenat cu anti-aliasing folosind un buffer de acumulare.

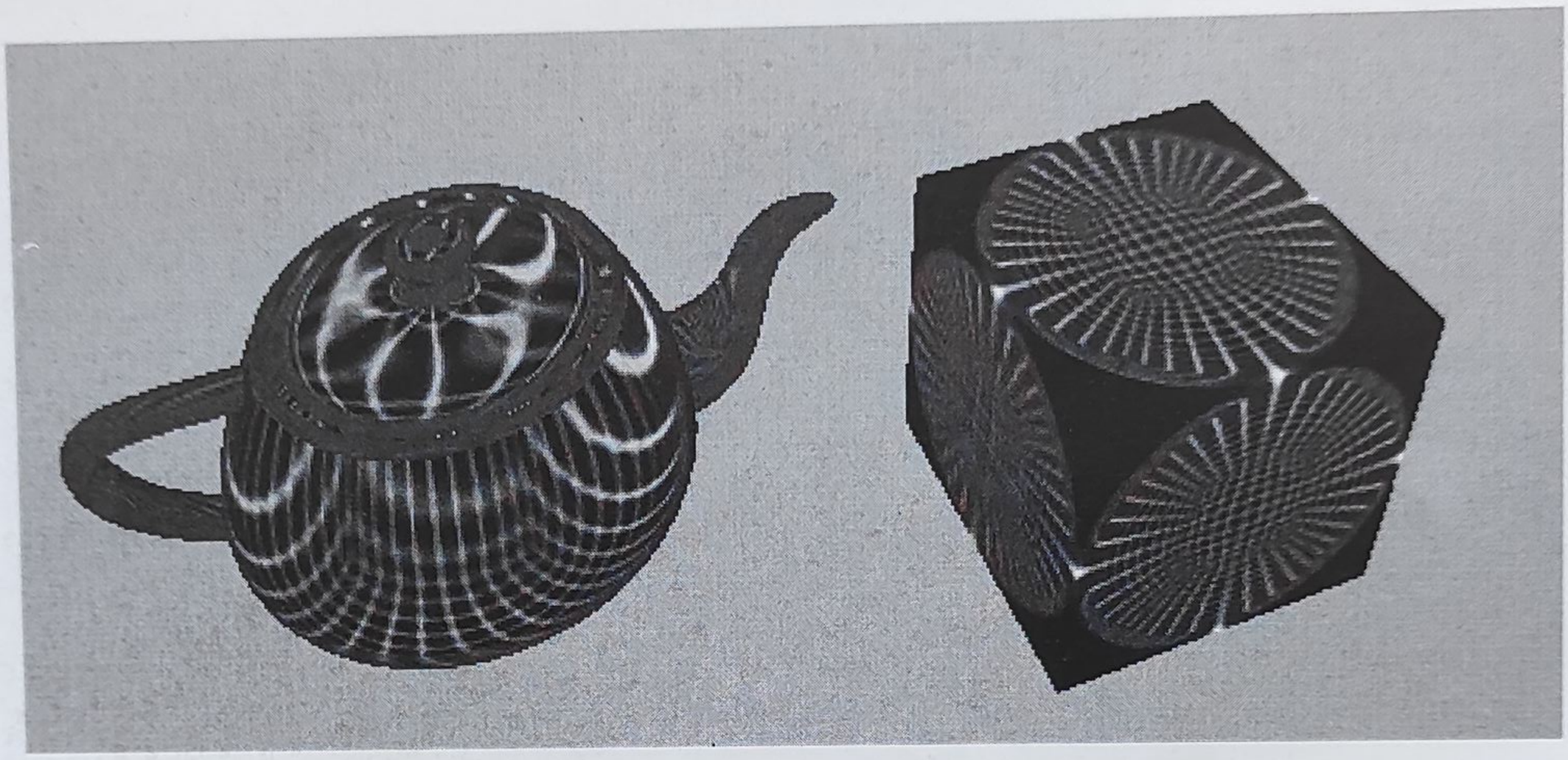


a) b)

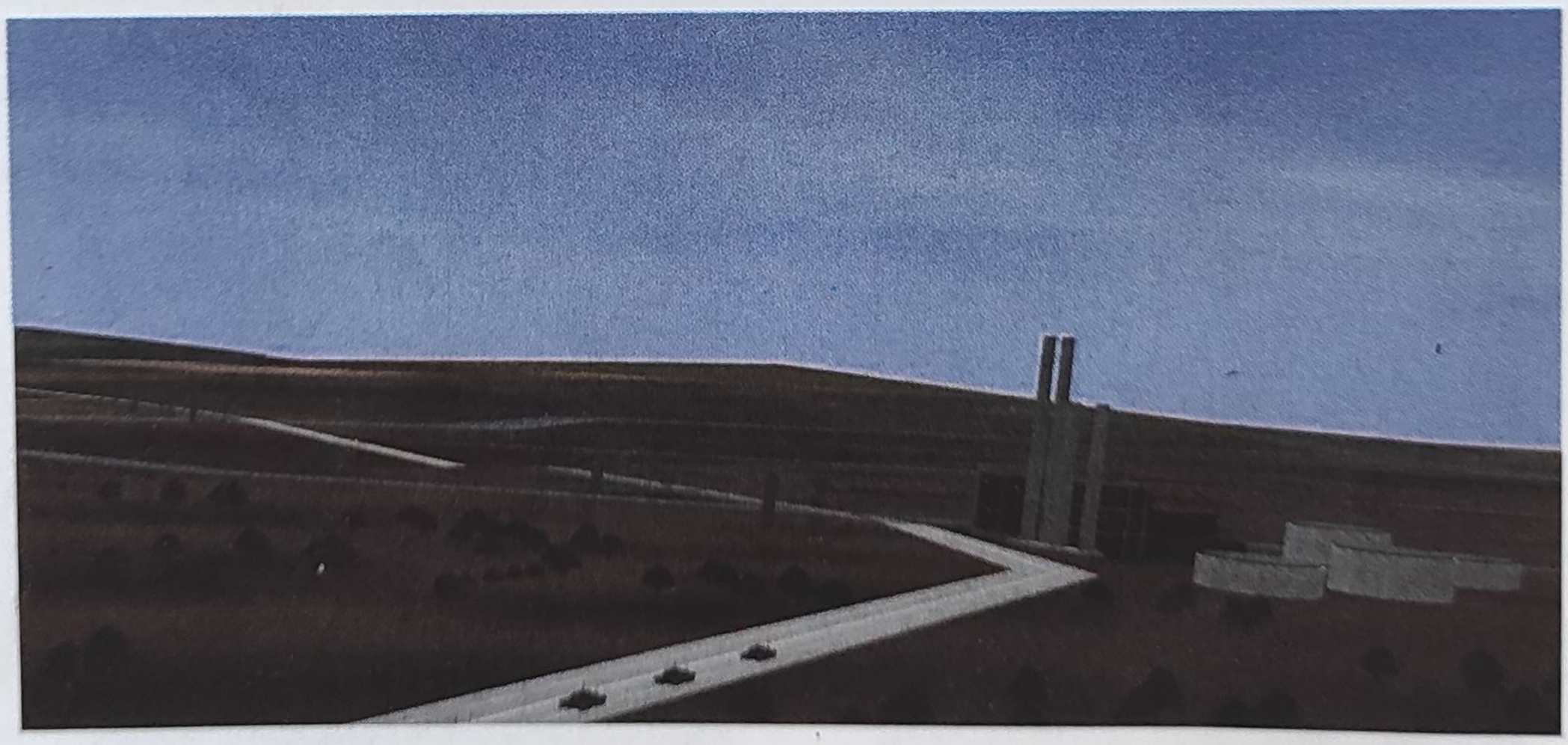
PLANŞA 7. Suprafețe ale obiectelor texturate prin aplicarea unei texturi bidimensionale:
a) aplicația repetată a imaginii unei cărămizi;
b) aplicația unei fotografii.



PLANŞA 8. a) Imagine cu textură nefiltrată. **b)** Imagine cu aceeaşi textură filtrată.



PLANŞA 9. Aplicația sferică a texturii pe obiectul „ceainic“ şi aplicația plană pe fiecare față a cubului.



PLANŞA 10. Imagine dintr-un simulator de zbor. Suprafețele sunt texturate, cu umbrire Gouraud şi anti-aliasing.

Prof. univ. dr. ing. FELICIA IONESCU este titular al disciplinelor *Sisteme Paralele şi Distribuite*, *Grafica în realitatea virtuală*, *Programare obiect-orientată* în cadrul Catedrei de Electronică Aplicată şi Ingineria Informaţiei din Facultatea de Electronică şi Telecomunicaţii, Universitatea "POLITEHNICA" Bucureşti. În activitatea de cercetare pe care a desfăşurat-o la *Institutul de Tehnică de Calcul* Bucureşti şi apoi la *Institutul de Simulatoare Simultec*, a coordonat activitatea de cercetare şi proiectare a mai multor sisteme de generare a imaginii, integrate în simulatoare de antrenament pentru piloţi.

Lucrarea **GRAFICA ÎN REALITATEA VIRTUALĂ** prezintă aspecte teoretice şi tehnici de programare a aplicaţiilor grafice, în special cele necesare în sistemele de realitate virtuală:

- Sisteme şi aplicaţii ale realităţii virtuale
- Modelarea obiectelor şi a scenelor virtuale
- Sisteme de vizualizare
- Modele de reflexie şi iluminare
- Tehnici de creare a realismului vizual: umbrire, anti-aliasing, texturare
- Exemple ilustrative de programare a aplicaţiilor grafice folosind biblioteca grafică OpenGL şi limbajul VRML

ISBN 973-31-1466-9